Національний технічний університет України
“Київський політехнічний інститут ім. Ігоря Сікорського”

Факультет прикладної математики
Кафедра системного програмування і спеціалізованих комп’ютерних систем

Розрахунково-графічна робота

з дисципліни

“Бази даних та засоби управління”

ТЕМА: “Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL”

Виконала: студентка ІІІ курсу
ФПМ групи КВ-21
Коноваленко Аліна
Оцінка:

Київ – 2024

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

Завдання роботи полягає у наступному:

1. Реалізувати функції 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).

Репозиторій GitHub
https://github.com/AlinaKonovalenko/databases_Konovalenko

Виконання роботи

Предметна галузь *«Електронний каталог для зберігання музичних нот та композицій»*

Опис сутностей предметної галузі

1. *Композиція* (Composition), з атрибутами: ID композиції, назва, тривалість, рік створення. Призначена для збереження інформації про музичні твори.
2. *Ноти* (Sheet Music), з атрибутами: ID нот, ID композиції, формат файлу, назва файлу. Призначена для збереження нотних записів музичних композицій.
3. *Автор* (Author), з атрибутами: ID автора, ім'я, прізвище. Призначена для збереження інформації про композиторів та аранжувальників.
4. *Жанр* (Genre), з атрибутами: ID жанру, назва жанру. Призначена для збереження інформації про категорії музичних творів.

Опис зв’язків муж сутностями предметної галузі

- Сутність "Композиція" має зв’язок 1:N по відношенню до сутності "Ноти", оскільки одна композиція може мати кілька нотних записів, але кожен нотний запис належить лише одній композиції.
- Сутність "Композиція" має зв’язок M:N по відношенню до сутності "Автор", оскільки одна композиція може мати кількох авторів, а один автор може створювати кілька композицій.

- Сутність "Композиція" має зв’язок N:1 по відношенню до сутності "Жанр", оскільки одна композиція належить лише до одного жанру, але до одного жанру можуть входити різні композиції.

Графічне подання концептуальної моделі «Сутність-зв’язок»

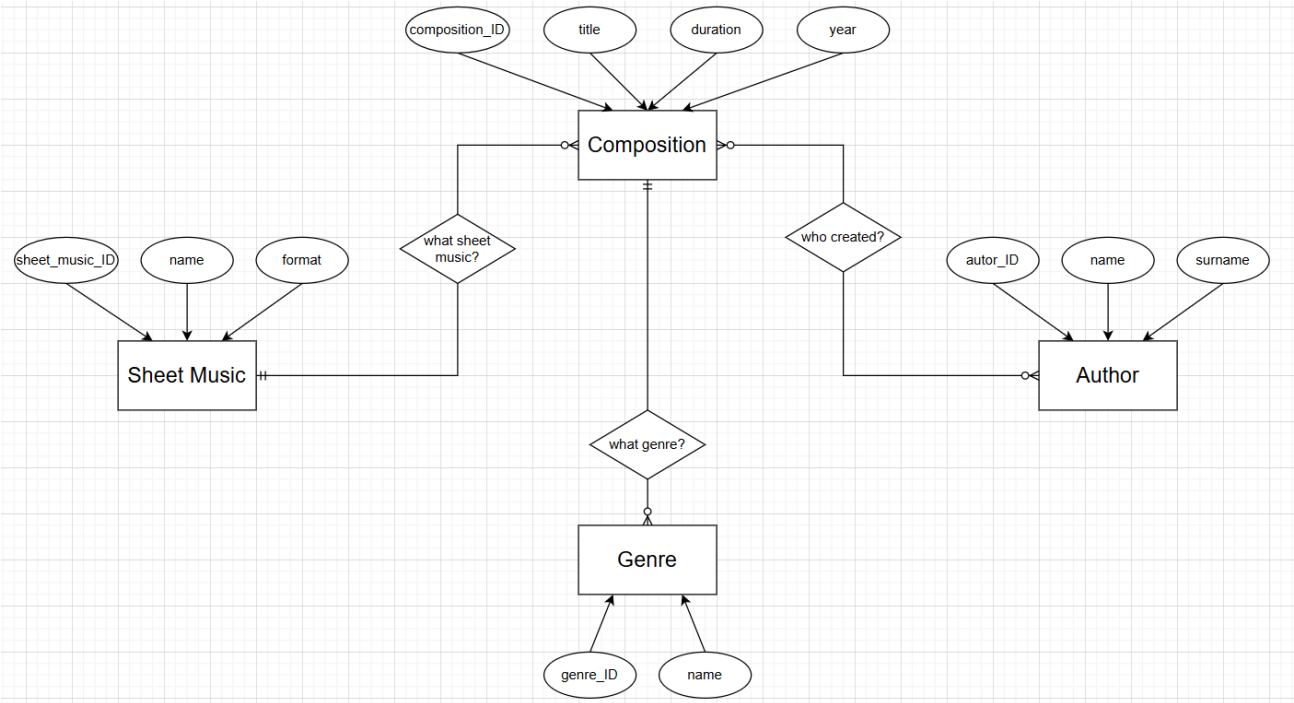


Рисунок 1 - ER-діаграма, побудована за нотацією Чена (інструмент: draw.io)

Графічне подання логічної моделі «Сутність-зв’язок»

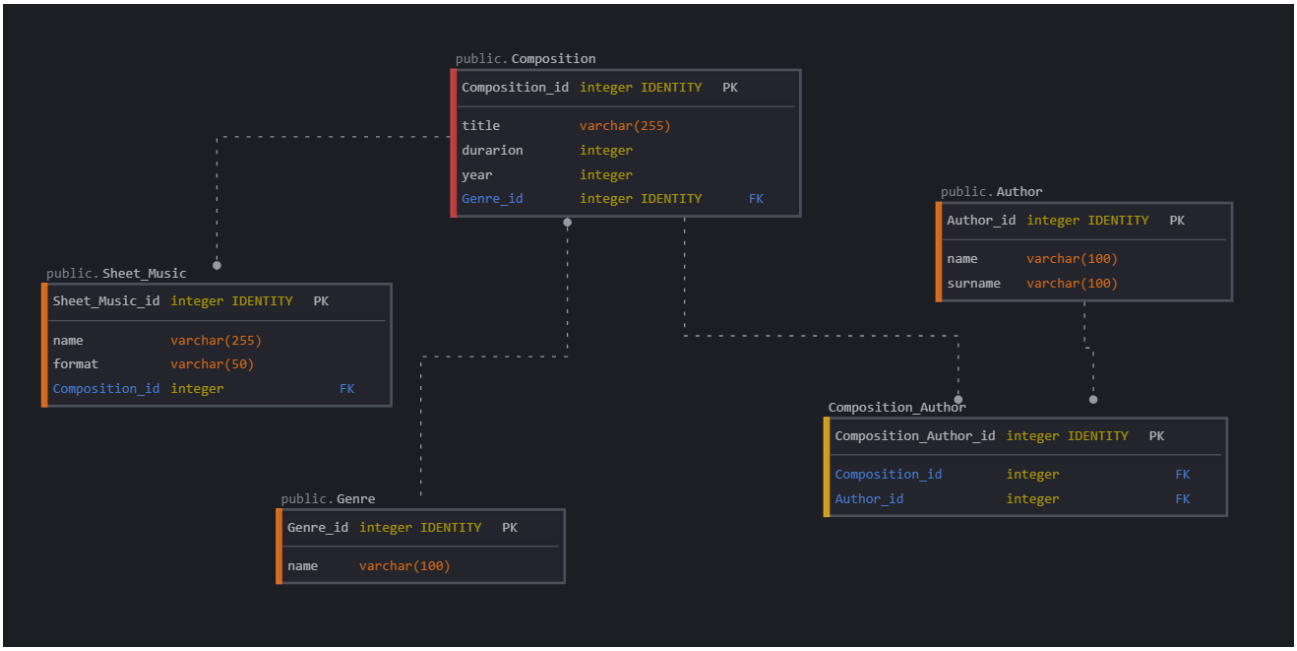


Рисунок 2 - Схема бази даних (інструмент: sqldbm.com)

Опис об’єктів бази даних

Сутність	*Атрибут*	*Тип атрибуту*
composition – містить інформацію про музичні композиції	composition_id – унікальний ідентифікатор композиції	integer (числовий)
	title – назва композиції	character varying (рядок)
	duration – тривалість композиції (у хвилинах)	integer (числовий)
	year – рік створення композиції	integer (числовий)
	genre_id – жанр до якого належить композиція	integer (числовий)
sheet_music – містить нотні записи композицій	sheet_music_id – унікальний ідентифікатор нот	integer (числовий)
	name – назва файлу нот	character varying (рядок)
	format – формат файлу нот	character varying (рядок)
	composition_id – композиція до якої належать ноти	integer (числовий)
author – містить інформацію про авторів	author_id – унікальний ідентифікатор автора	integer (числовий)
	name – ім’я автора	character varying (рядок)
	surname – прізвище автора	character varying (рядок)
genre – містить інформацію про жанри композицій	genre_id – унікальний ідентифікатор жанру	integer (числовий)
	name – назва жанру	character varying (рядок)
composition_author – зв’язує авторів із композиціями	composition_author_id – унікальний ідентифікатор зв’язку	integer (числовий)
	composition_id – унікальний ідентифікатор композиції	integer (числовий)
	author_id – унікальний ідентифікатор автора	integer (числовий)

Середовище та компоненти розробки

Для розробки використовувалась мова програмування C#, середовище розробки Visual Studio, а також стороння бібліотека, що надає API для доступу до PostgreSQL – Npgsql.

Шаблон проектування

MVC - шаблон проектування, який використаний у програмі.

- *Model* (Модель) - відповідає за логіку роботи з даними. Вона містить класи, які безпосередньо взаємодіють із базою даних, виконують операції додавання, оновлення, вилучення та отримання даних.
- *View* (Представлення) - відповідає за відображення даних для користувача. У нашому випадку це консольний інтерфейс, через який користувач взаємодіє з програмою.
- *Controller* (Контролер) - відповідає за обробку запитів користувача, взаємодію з моделлю та представленням. Він приймає вхідні дані від користувача, обробляє їх і передає результат у вигляді відповідного представлення.

Структура програми та її опис

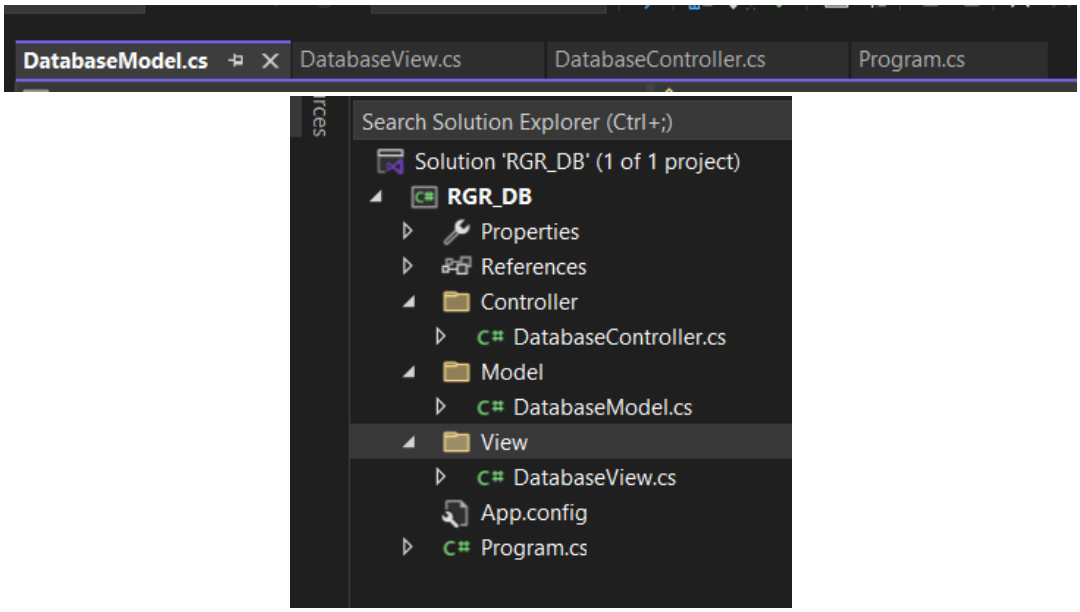


Рисунок 3 – Структура програми

Програма умовно розділена на 4 модулі: файл DatabaseController.cs, файл DatabaseModel.cs, файл DatabaseView.cs та головний файл Program.cs. Класи, як видно з їхніх назв, повністю відповідають використаному патерну MVC.

- У файлі *DatabaseModel* описаний клас моделі, що займається регулюванням підключення до бази даних, та виконанням запитів до неї.

- У файлі *DatabaseView* описаний клас, що виводить результати виконання тієї чи іншої дії на екран консолі.
- У файлі *DatabaseController* описаний інтерфейс взаємодії з користувачем, запит бажаної дії, виконання пошуку, тощо.
- *Program* – головний файл, що запускає програму.

Структура меню програми

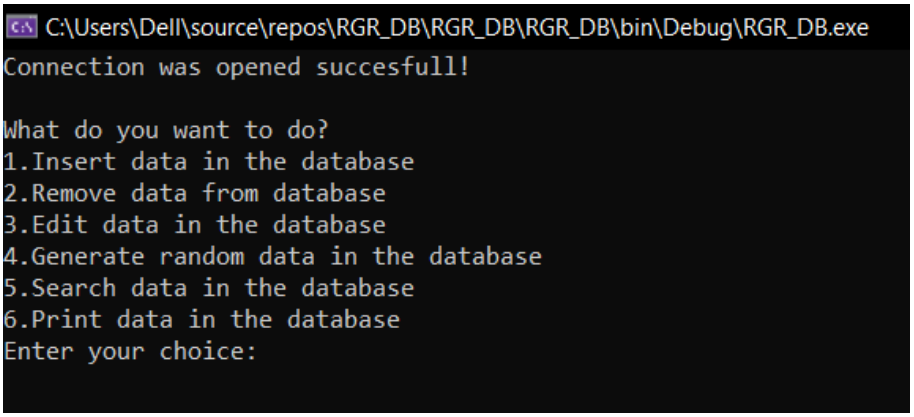


Рисунок 4 – Меню програми

Меню користувача складається з шести варіантів вибору:

1. Перший варіант дозволяє ввести дані до таблиці.
2. Другий варіант дає можливість видалити дані з таблиці.
3. Третій варіант забезпечує оновлення даних у таблиці.
4. Четвертий варіант дозволяє генерувати дані для таблиці.
5. П’ятий варіант призначений для пошуку даних у таблиці.
6. Шостий варіант надає інформацію про імена та типи стовпців таблиці.

Фрагменти програм внесення, редагування та вилучення даних у БД:

- Введення даних в таблицю

```
    public void Insert() {
            List<string> tables = GetTables("insert");
            int selected_index = int.Parse(Console.ReadLine()) - 1;
            List<string> columns = GetColumns(tables[selected_index]);
            List<string> values = new List<string>();
            foreach (string column in columns) {
                Console.Write("Enter value for column - " + column + " - ");
                values.Add(Console.ReadLine());
            }
            string insert_query = "INSERT INTO " + tables[selected_index] + "
(" + ListToString(columns, false) + ") VALUES (" + ListToString(values, true)
+ ")";
            NpgsqlCommand insert_command = new NpgsqlCommand(insert_query,
connection);
            Console.WriteLine(insert_query);
            insert_command.ExecuteNonQuery();
            Console.WriteLine("Data succesfull added!");
```

```
        }
```

- Видалення даних з таблиці

```
        public void Delete() {
            List<string> tables = GetTables("delete");
            int selected_index = int.Parse(Console.ReadLine()) - 1;
            Console.WriteLine("Input row id for delete");
            int row_id = int.Parse(Console.ReadLine());
            NpgsqlCommand delete_command = new NpgsqlCommand("DELETE FROM " +
tables[selected_index] + " WHERE " + tables[selected_index] + "_id = " +
row_id, connection);
            delete_command.ExecuteNonQuery();
            Console.WriteLine("Data succesfull deleted!");
        }
```

- Оновлення даних в таблиці

```
    public void Update() {
            List<string> tables = GetTables("update");
            int selected_index = int.Parse(Console.ReadLine()) - 1;
            List<string> columns = GetColumns(tables[selected_index]);
            List<string> values = new List<string>();
            Console.WriteLine("Input row id for update");
            int row_id = int.Parse(Console.ReadLine());
            foreach (string column in columns) {
                Console.Write("Enter new value for column - " + column + " -
");
                values.Add(Console.ReadLine());
            }
            string update_query = "UPDATE " + tables[selected_index] + " SET
" + UpdatePartialString(columns, values) + " WHERE " + tables[selected_index]
+ "_id = " + row_id;
            NpgsqlCommand update_command = new NpgsqlCommand(update_query,
connection);
            update_command.ExecuteNonQuery();
            Console.WriteLine("Data succesfull updated! " + update_query);
        }
```

- Генерування даних в таблиці

```
          public void Generate() {
              List<string> tables = GetTables("generate");
              int selected_index = int.Parse(Console.ReadLine()) - 1;
              Console.WriteLine("Input rows count for generate");
              int rows_count = int.Parse(Console.ReadLine());
              List<string> columns = GetColumns(tables[selected_index]);
              Random random = new Random();
              for (int i = 0; i < rows_count; i++) {
                  while (true) {
                      try {
                          List<string> values = new List<string>();
                          foreach (string column in columns) {
                              values.Add(random.Next(0, 10).ToString());
                          }
                          //values[values.Count - 1] = "1";
                          string insert_query = "INSERT INTO " +
   tables[selected_index] + " (" + ListToString(columns, false) + ") VALUES
   (" + ListToString(values, true) + ")";
                          NpgsqlCommand insert_command = new
   NpgsqlCommand(insert_query, connection);
                          insert_command.ExecuteNonQuery();
```

```
                        break;
                    }
                    catch (Exception ex) {
                        //Console.WriteLine(ex.Message);
                    }
                }
            }
            Console.WriteLine("Generated was succesfull!");
        }
```

- Пошук даних в таблиці

```
    public void Search()
{
    Console.WriteLine("Choose what to search:");
    Console.WriteLine("1 - Select compositions by genre_id");
    Console.WriteLine("2 - Select authors by composition_id");
    Console.WriteLine("3 - Select compositions by author_id");
    Console.Write("Your choice: ");
    int search = int.Parse(Console.ReadLine());

    switch (search)
    {
        case 1:
            Console.Write("Input genre_id: ");
            string genre_id = Console.ReadLine();
            ExecuteSearchQuery("SELECT title FROM composition WHERE genre_id
= @genre_id", genre_id);
            break;

        case 2:
            Console.Write("Input composition_id: ");
            string composition_id = Console.ReadLine();
            ExecuteSearchQuery("SELECT a.name, a.surname FROM author a JOIN
composition_author ca ON a.author_id = ca.author_id WHERE ca.composition_id =
@composition_id", composition_id);
            break;

        case 3:
            Console.Write("Input author_id: ");
            string author_id = Console.ReadLine();
            ExecuteSearchQuery("SELECT c.title FROM composition c JOIN
composition_author ca ON c.composition_id = ca.composition_id WHERE
ca.author_id = @author_id", author_id);
            break;

        default:
            Console.WriteLine("Invalid choice!");
            break;
    }
}

private void ExecuteSearchQuery(string query, string parameterValue)
{
    try
    {
        NpgsqlCommand command = new NpgsqlCommand(query, connection);
        command.Parameters.AddWithValue("@parameter", parameterValue);

        NpgsqlDataReader reader = command.ExecuteReader();
```

```
        List<string> data = new List<string>();

        while (reader.Read())
        {
            data.Add(reader.GetValue(0).ToString());
        }

        if (data.Count > 0)
        {
            PrintRow(data);
        }
        else
        {
            Console.WriteLine("No results found.");
        }

        reader.Close();
    }
    catch (Exception ex)
    {
        Console.WriteLine("An error occurred: " + ex.Message);
    }
}
```

- Отримання імен та типів стовпчиків таблиці

```
     public void Print() {
            List<string> tables = GetTables("print");
            int selected_index = int.Parse(Console.ReadLine()) - 1;
            List<string> columns = GetColumns(tables[selected_index]);
            PrintRow(columns);
            NpgsqlCommand command = new NpgsqlCommand("SELECT * FROM " +
tables[selected_index], connection);
            NpgsqlDataReader reader = command.ExecuteReader();
            List<string> data = new List<string>();
            while (reader.Read()) {
                for (int i = 0; i < columns.Count; i++) {
                    data.Add(reader.GetValue(i).ToString());
                }
                PrintRow(data);
                data.Clear();
            }
            reader.Close();
        }
```

Наведені вище фрагменти програми відповідають за функціонал додавання, редагування та видалення даних у базі даних. Взаємодія з базою здійснюється через клас Model, який відповідає за підключення до БД, а самі функції реалізовані в файлі Controller.

Результати виконання програми

Виконання операції вставки запису в таблицю

```
Microsoft Visual Studio Debug Console
Connection was opened succesfull!

What do you want to do?
1.Insert data in the database
2.Remove data from database
3.Edit data in the database
4.Generate random data in the database
5.Search data in the database
6.Print data in the database
Enter your choice: 1
Choose table to insert data
1.author
2.composition
3.composition_author
4.genre
5.sheet_music
4
Enter value for column - genre_id - 4
Enter value for column - name - blues
INSERT INTO genre (name) VALUES ('blues')
```

Data Output | Messages | Notifications

	genre_id [PK] integer	name character varying (100)
1	1	classical
2	2	rock
3	3	pop

Data Output | Messages | Notifications

	genre_id [PK] integer	name character varying (100)
1	1	classical
2	2	rock
3	3	pop
4	4	blues

Виконання операції видалення даних з таблиці

```
C:\Users\Dell\source\repos\RGR_DB\RGR_DB\RGR_
Connection was opened succesfull!

What do you want to do?
1.Insert data in the database
2.Remove data from database
3.Edit data in the database
4.Generate random data in the database
5.Search data in the database
6.Print data in the database
Enter your choice: 2
Choose table to delete data
1.author
2.composition
3.composition_author
4.genre
5.sheet_music
4
Input row id for delete
4
Data succesfull deleted!
```

Data Output | Messages | Notifications

	genre_id [PK] integer	name character varying (100)
1	1	classical
2	2	rock
3	3	pop
4	4	blues

Data Output | Messages | Notifications

	genre_id [PK] integer	name character varying (100)
1	1	classical
2	2	rock
3	3	pop

Виконання операції оновлення даних в таблиці

```
What do you want to do?
1.Insert data in the database
2.Remove data from database
3.Edit data in the database
4.Generate random data in the database
5.Search data in the database
6.Print data in the database
Enter your choice: 3
Choose table to update data
1.author
2.composition
3.composition_author
4.genre
5.sheet_music
5
Input row id for update
1
Enter new value for column - sheet_music_id - 1
Enter new value for column - name - Cello Solo
Enter new value for column - format - Audio
Enter new value for column - composition_id - 1
Data succesfull updated! UPDATE sheet_music SET name = 'Cello Solo', format = 'Audio', composition_id = '1' WHERE sheet_music_id = 1
```

Data Output | Messages | Notifications

	sheet_music_id [PK] integer	name character varying (255)	format character varying (50)	composition_id integer
0+	1	Cello Concerto No.1 in C major, 1st Mvt. Solo	PDF	1
1+	2	Cello Concerto C Major Mvt. 1 (Piano Accompaniment)	PDF	1
2+	3	I'm still standing	MIDI	3
3+	4	Starman	Audio	2

Data Output | Messages | Notifications

	sheet_music_id [PK] integer	name character varying (255)	format character varying (50)	composition_id integer
1	1	Cello Solo	Audio	1
2	2	Cello Concerto C Major Mvt.1 (Piano Accompaniment)	PDF	1
3	3	I'm staill standing	MIDI	3
4	4	Starman	Audio	2

Виконання операції генерування даних в таблиці

```
C:\Users\Dell\source\repos\RGR_DB\RGR_DB\RGR_
Connection was opened succesfull!

What do you want to do?
1.Insert data in the database
2.Remove data from database
3.Edit data in the database
4.Generate random data in the database
5.Search data in the database
6.Print data in the database
Enter your choice: 4
Choose table to generate data
1.author
2.composition
3.composition_author
4.genre
5.sheet_music
1
Input rows count for generate
3
```

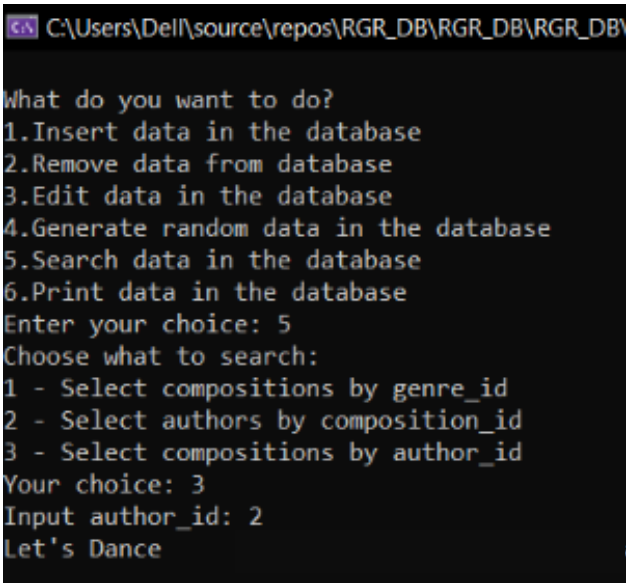
Data Output | Messages | Notifications

	author_id [PK] integer	name character varying (100)	surname character varying (100)
1	1	Joseph	Haydn
2	2	David	Bowie
3	3	Elton	John

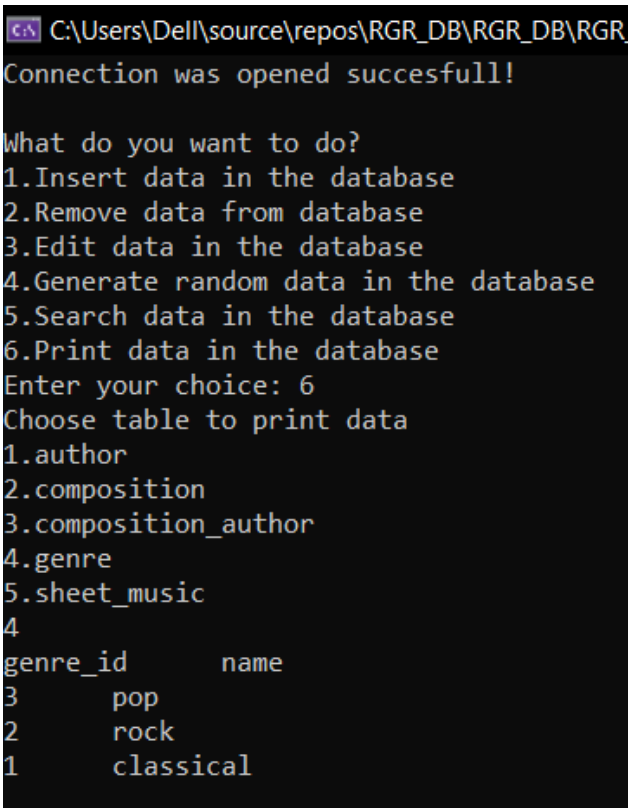
Data Output | Messages | Notifications

	author_id [PK] integer	name character varying (100)	surname character varying (100)
1	1	Joseph	Haydn
2	2	David	Bowie
3	3	Elton	John
4	4	17	19
5	5	5	16
6	6	3	83

Виконання операції пошуку даних в таблиці



Виконання операції отримання імен та типів стовпчиків таблиці



Код програми

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Npgsql;

namespace PostgreConsoleInteractorCS
{
    class Program
    {
        static void Main(string[] args)
        {
            DatabaseView view = new DatabaseView();
            view.Process();
        }
    }
}
```

DatabaseView.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace PostgreConsoleInteractorCS
{
    public class DatabaseView
    {
        DatabaseController controller;

        public void Process()
        {
            string connection_string = "Server=localhost;Port=5432;User
ID=postgres;Password=0000;Database=postgres;";
            controller = new DatabaseController(connection_string);
            while (true)
            {
                int action = controller.ShowActionList();
                controller.Execute(action);
            }
        }
    }
}
```

DatabaseController.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;
using Npgsql;
namespace PostgreConsoleInteractorCS
{
    public class DatabaseController
    {
        DatabaseModel db = null;
        public DatabaseController(string connection_string)
        {
            db = new DatabaseModel(connection_string);
        }

        public int ShowActionList()
        {
            int choose;
            Console.WriteLine("\nWhat do you want to do?");
            Console.WriteLine("1.Insert data in the database");
            Console.WriteLine("2.Remove data from database");
            Console.WriteLine("3.Edit data in the database");
            Console.WriteLine("4.Generate random data in the database");
            Console.WriteLine("5.Search data in the database");
            Console.WriteLine("6.Print data in the database");
            Console.Write("Enter your choice: ");
            choose = int.Parse(Console.ReadLine());
            return choose;
        }

        public void Execute(int action)
        {
            if (db != null)
            {
                switch (action)
                {
                    case 1:
                        db.Insert();
                        break;
                    case 2:
                        db.Delete();
                        break;
                    case 3:
                        db.Update();
                        break;
                    case 4:
                        db.Generate();
                        break;
                    case 5:
                        db.Search();
                        break;
                    case 6:
                        db.Print();
                        break;
                    default:
                        Console.WriteLine("Not correct operation. Try again!");
                        break;
                }
            }
        }
    }
}
```

DatabaseModel.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Npgsql;

namespace PostgreConsoleInteractorCS
{
    public class DatabaseModel
    {
        NpgsqlConnection connection;
        public DatabaseModel(string connection_string)
        {
            connection = new NpgsqlConnection(connection_string);
            connection.Open();
            Console.WriteLine("Connection was opened succesfull!");
        }

        public void Insert()
        {
            List<string> tables = GetTables("insert");
            int selected_index = int.Parse(Console.ReadLine()) - 1;
            List<string> columns = GetColumns(tables[selected_index]);
            List<string> values = new List<string>();
            foreach (string column in columns)
            {
                Console.Write("Enter value for column - " + column + " - ");
                values.Add(Console.ReadLine());
            }
            string insert_query = "INSERT INTO " + tables[selected_index] + " ("
+ ListToString(columns, false) + ") VALUES (" + ListToString(values, true) +
")";
            NpgsqlCommand insert_command = new NpgsqlCommand(insert_query,
connection);
            Console.WriteLine(insert_query);
            insert_command.ExecuteNonQuery();
            Console.WriteLine("Data succesfull added!");
        }

        public void Delete()
        {
            List<string> tables = GetTables("delete");
            int selected_index = int.Parse(Console.ReadLine()) - 1;
            Console.WriteLine("Input row id for delete");
            int row_id = int.Parse(Console.ReadLine());
            NpgsqlCommand delete_command = new NpgsqlCommand("DELETE FROM " +
tables[selected_index] + " WHERE " + tables[selected_index] + "_id = " + row_id,
connection);
            delete_command.ExecuteNonQuery();
            Console.WriteLine("Data succesfull deleted!");
        }

        public void Update()
        {
            List<string> tables = GetTables("update");
            int selected_index = int.Parse(Console.ReadLine()) - 1;
```

```
            List<string> columns = GetColumns(tables[selected_index]);
            List<string> values = new List<string>();
            Console.WriteLine("Input row id for update");
            int row_id = int.Parse(Console.ReadLine());
            foreach (string column in columns)
            {
                Console.Write("Enter new value for column - " + column + " - ");
                values.Add(Console.ReadLine());
            }
            string update_query = "UPDATE " + tables[selected_index] + " SET " +
UpdatePartialString(columns, values) + " WHERE " + tables[selected_index] + "_id
= " + row_id;
            NpgsqlCommand update_command = new NpgsqlCommand(update_query,
connection);
            update_command.ExecuteNonQuery();
            Console.WriteLine("Data succesfull updated! " + update_query);
        }

        public void Generate()
        {
            List<string> tables = GetTables("generate");
            int selected_index = int.Parse(Console.ReadLine()) - 1;
            Console.WriteLine("Input rows count for generate");
            int rows_count = int.Parse(Console.ReadLine());
            List<string> columns = GetColumns(tables[selected_index]);
            Random random = new Random();
            for (int i = 0; i < rows_count; i++)
            {
                while (true)
                {
                    try
                    {
                        List<string> values = new List<string>();
                        foreach (string column in columns)
                        {
                            values.Add(random.Next(0, 10).ToString());
                        }
                        //values[values.Count - 1] = "1";
                        string insert_query = "INSERT INTO " +
tables[selected_index] + " (" + ListToString(columns, false) + ") VALUES (" +
ListToString(values, true) + ")";
                        NpgsqlCommand insert_command = new
NpgsqlCommand(insert_query, connection);
                        insert_command.ExecuteNonQuery();
                        break;
                    }
                    catch (Exception ex)
                    {
                        //Console.WriteLine(ex.Message);
                    }
                }
            }
            Console.WriteLine("Generated was succesfull!");
        }

        public void Search()
        {
            Console.WriteLine("Choose what to search:");
            Console.WriteLine("1 - Select compositions by genre_id");
```

```
            Console.WriteLine("2 - Select authors by composition_id");
            Console.WriteLine("3 - Select compositions by author_id");
            Console.Write("Your choice: ");
            int search = int.Parse(Console.ReadLine());

            switch (search)
            {
                case 1:
                    Console.Write("Input genre_id: ");
                    string genre_id = Console.ReadLine();
                    ExecuteSearchQuery("SELECT title FROM composition WHERE
genre_id = @genre_id", genre_id);
                    break;

                case 2:
                    Console.Write("Input composition_id: ");
                    string composition_id = Console.ReadLine();
                    ExecuteSearchQuery("SELECT a.name, a.surname FROM author a
JOIN composition_author ca ON a.author_id = ca.author_id WHERE ca.composition_id
= @composition_id", composition_id);
                    break;

                case 3:
                    Console.Write("Input author_id: ");
                    string author_id = Console.ReadLine();
                    ExecuteSearchQuery("SELECT c.title FROM composition c JOIN
composition_author ca ON c.composition_id = ca.composition_id WHERE ca.author_id
= @author_id", author_id);
                    break;

                default:
                    Console.WriteLine("Invalid choice!");
                    break;
            }
        }

        private void ExecuteSearchQuery(string query, string parameterValue)
        {
            try
            {
                NpgsqlCommand command = new NpgsqlCommand(query, connection);
                command.Parameters.AddWithValue("@parameter", parameterValue);

                NpgsqlDataReader reader = command.ExecuteReader();
                List<string> data = new List<string>();

                while (reader.Read())
                {
                    data.Add(reader.GetValue(0).ToString());
                }

                if (data.Count > 0)
                {
                    PrintRow(data);
                }
                else
                {
                    Console.WriteLine("No results found.");
                }
```

```
                reader.Close();
            }
            catch (Exception ex)
            {
                Console.WriteLine("An error occurred: " + ex.Message);
            }
        }


        public void Print()
        {
            List<string> tables = GetTables("print");
            int selected_index = int.Parse(Console.ReadLine()) - 1;
            List<string> columns = GetColumns(tables[selected_index]);
            PrintRow(columns);
            NpgsqlCommand command = new NpgsqlCommand("SELECT * FROM " +
tables[selected_index], connection);
            NpgsqlDataReader reader = command.ExecuteReader();
            List<string> data = new List<string>();
            while (reader.Read())
            {
                for (int i = 0; i < columns.Count; i++)
                {
                    data.Add(reader.GetValue(i).ToString());
                }
                PrintRow(data);
                data.Clear();
            }
            reader.Close();
        }

        private List<string> GetTables(string operation)
        {
            Console.WriteLine("Choose table to " + operation + " data");
            NpgsqlCommand command = new NpgsqlCommand("SELECT * FROM
information_schema.tables WHERE table_schema = 'public';", connection);
            NpgsqlDataReader reader = command.ExecuteReader();
            List<string> tables = new List<string>();
            while (reader.Read())
            {
                tables.Add(reader.GetString(2));
            }
            reader.Close();
            int index = 1;
            foreach (string table in tables)
            {
                Console.WriteLine(index.ToString() + '.' + table);
                index++;
            }
            return tables;
        }

        private List<string> GetColumns(string table)
        {
            string query = "SELECT * FROM information_schema.columns WHERE
table_schema = 'public' AND table_name = '" + table + "'";
            NpgsqlCommand command = new NpgsqlCommand(query, connection);
            NpgsqlDataReader reader = command.ExecuteReader();
```

```
            List<string> columns = new List<string>();
            while (reader.Read())
            {
                columns.Add(reader.GetString(3));
            }
            reader.Close();
            return columns;
        }

        private string ListToString(List<string> list, bool is_value)
        {
            string result = string.Empty;
            for (int i = 1; i < list.Count; i++)
            {
                if (is_value) result += "'" + list[i] + "',";
                else result += list[i] + ',';
            }
            result = result.Remove(result.Length - 1, 1);
            return result;
        }

        private void PrintRow(List<string> list)
        {
            foreach (string s in list)
            {
                Console.Write(s + "      ");
            }
            Console.Write('\n');
        }

        private string UpdatePartialString(List<string> columns, List<string>
values)
        {
            string partial = string.Empty;
            for (int i = 1; i < columns.Count; i++)
            {
                partial += columns[i] + " = '" + values[i] + "'";
                if (i < columns.Count - 1) partial += ", ";
            }
            return partial;
        }
    }
}
```

Опис функцій модуля DatabaseModel

DatabaseModel - Клас для взаємодії з базою даних PostgreSQL. Конструктор встановлює з'єднання з базою даних за допомогою переданого рядка підключення.

- *Insert* - Додає нові записи в обрану таблицю. Користувач вводить значення для кожного стовпця. Формується SQL-запит для вставки даних.
- *Delete* - Видаляє запис із вибраної таблиці за вказаним ідентифікатором. Видалення здійснюється за унікальним ідентифікатором рядка.
- *Update* - Оновлює значення у вибраному рядку таблиці. Користувач вводить нові значення для всіх стовпців рядка за його ідентифікатором.
- *Generate* - Генерує випадкові записи для вибраної таблиці. Користувач вказує кількість рядків для генерації. Дані генеруються випадковим чином.
- *Search* - Виконує пошукові запити на основі вибраних критеріїв: композиції за жанром, автори за композицією, композиції за автором.
- *ExecuteSearchQuery* - Використовується для виконання SQL-запитів у методі пошуку. Формує параметризований запит і виводить результати.
- *Print* - Виводить усі записи вибраної таблиці. Зчитує всі стовпці та їхні значення, формуючи таблицю для перегляду.
- *GetTables* - Повертає список таблиць у базі даних. Відображає користувачеві доступні таблиці для вибору.
- *GetColumns* - Повертає список стовпців для вибраної таблиці. Використовується в методах для генерації запитів.
- *ListToString* - Перетворює список у рядок для формування SQL-запиту. Додає лапки для значень, якщо це потрібно.
- *PrintRow* - Виводить окремий рядок даних у вигляді таблиці.
- *UpdatePartialString* - Формує частину SQL-запиту для оновлення даних. З’єднує назви стовпців і нові значення в форматі 'колонка = значення'.

Призначення модуля

Модуль забезпечує операції створення, читання, оновлення, видалення та генерацію випадкових даних для роботи з базами даних PostgreSQL через консольний інтерфейс.