A black and white drawing of a building

Description automatically generated

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**

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

Виконав:

студент ІІI курсу

групи КВ-23

Марінченко М. О.

Київ – 2024  
**Мета:** здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

**Виконання роботи**

**Сутності предметної області**

1. Клієнт(Client), з атрибутами: код клієнта, ім’я, прізвище,електронна пошта, номер телефона, дата реєстрації. Призначена для збереження інформації про клієнта;

2. Бронювання (Bookings), з атрибутами: код бронювання, код клієнта, код сесії, дата бронювання, статус бронювання. Призначена для збереження інформації про бронювання відповідним клієнтом на відповідну сесію.

3. Сесія (Sessions), з атрибутами: код сесії, код тренера, код локації, час початку, час закінчення, максимальна кількість учасників, ціна. Призначена для збереження інформації про сесію тренування;

4. Локації (Locations), з атрибутами: код локації, назва, адресса, місто, вмістивність. Призначена для збереження інформації про місце проведення тренування.

5. Тренери(Instructors), з атрибутами: код тренера, досвід роботи, опис(біографія), рейтинг. Призначена для збереження інформації про тренерів.

**Зв’язки між сутностями предметної області**

Сутність “Клієнт” має зв’язок 1:N по відношенню до сутності “Бронювання ” оскільки клієнт може містити декілька бронювань.

Сутність “Сесія” має зв'язок 1:N по відношенню до “Бронювань”, оскільки одна сесія може бути заброньованна декілька разів.

Сутність “Локація” має зв'язок 1:N по відношенню до “Сесія”, оскільки одна локація може бути місцем проведенням для багатьох сесій, але сесія має лише одне місце проведення.

Сутність “Тренер” має зв’язок 1:N по відношенню до “Сесія”, оскільки один тренер може проводити декілька тренувальних сесій, але у сесії завжди один тільки тренер.

Графічне подання логічної моделі «Сутність-зв’язок» зображено на рисунку 1.

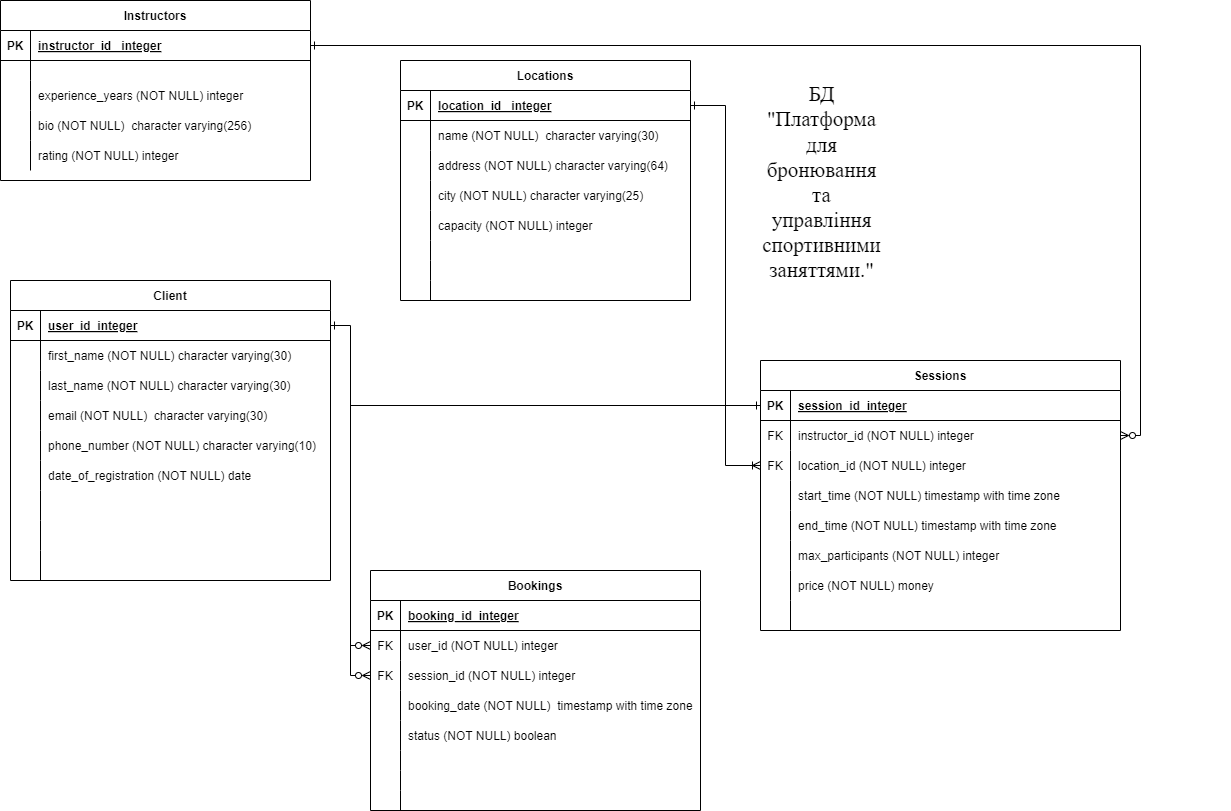


Рисунок 1 – Логічна модель

**Середовище та компоненти розробки**

У процесі розробки була використана мова програмування C#, інтегроване середовище розробки Visual Studio 2022, а також була використана бібліотека Npgsql v8.0.5, яка надає API для взаємодії з базою даних PostgreSQL.

**Шаблон проектування**

Модель-представлення-контролер (MVC) – це шаблон проектування, що використовується у програмі. Кожен компонент відповідає за певну функціональну частину:

1. Модель (Model) – це клас, що відображає логіку роботи з даними, обробляє всі операції з даними, такі як додавання, оновлення, вилучення.
2. Представлення (View) – це клас, через який користувач взаємодіє з програмою. У даному випадку, консольний інтерфейс, який відображає дані для користувача та зчитує їх з екрану.
3. Контролер (Controller) – це клас, який відповідає за зв’язок між користувачем і системою. Він приймає введені користувачем дані та обробляє їх. В залежності від результатів, викликає відповідні дії з Model або View.

Даний підхід дозволяє розділити логіку програми на логічні компоненти, що полегшує розробку, тестування і підтримку продукту.

**Структура програми та її опис**

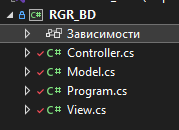


Рисунок 2 – Структура програми

З файлу Program.cs відбувається виклик контролера та передача йому управління.

У файлі Model.cs описаний клас моделі, який відповідає за управління підключенням до бази даних і виконанням низькорівневих запитів до неї.

У файлі Controller.cs реалізовано інтерфейс взаємодії з користувачем, включаючи обробку запитів користувача, а також інші дії, необхідні для взаємодії з моделлю та представленням.

У файлі View.cs описаний клас, який відображає результати виконання різних дій користувача на екрані консолі. Цей компонент відповідає за представлення даних користувачу в зручному для сприйняття вигляді.

Отже, структура програми відповідає патерну MVC.

**Структура меню програми**

На рисунку 3 зображено меню користувача, яке складається з семи пунктів.

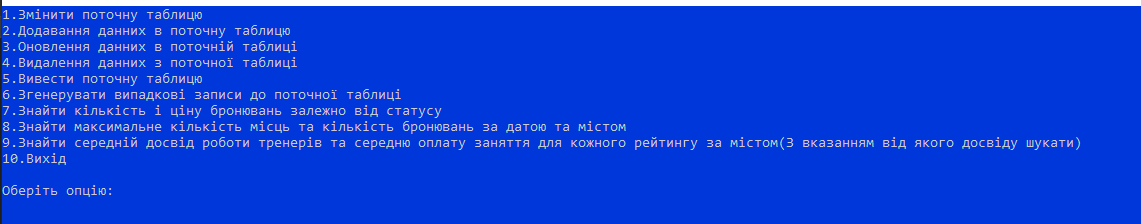


Рисунок 3 – Структура меню користувача

**Фрагмент коду (файл Controller.cs), в якому наведено головний цикл роботи програми**

public static void Run()

{

bool cycle\_run = true;

while (cycle\_run)

{

int choice = view.ShowMenu();

if (current\_table == "None" && choice < 7 && choice != 1)

{

Console.WriteLine("Оберіть таблицю для роботи!!!");

Thread.Sleep(1000);

}

else

{

switch (choice)

{

case 1:

ChangeCurTable();

break;

case 2:

AddDataToTable();

break;

case 3:

UpdateTableData();

break;

case 4:

DeleteData();

break;

case 5:

ShowCurTable();

break;

case 6:

GenerateData();

break;

case 7:

Search(1);

break;

case 8:

Search(2);

break;

case 9:

Search(3);

break;

case 10:

cycle\_run = !cycle\_run;

break;

}

}

}

}

**Фрагмент коду (файл Model.cs), в якому наведено функції внесення, редагування, вилучення та генерації даних у базі даних**

Функція внесення даних:

public bool AddDataToTableModel(List<(string Column, string Value)> values,string table\_name)

{

try

{

connection.Open();

}

catch (Exception ex)

{

Console.WriteLine("Помилка при підключенні до бази данних");

return true;

}

Dictionary<string, string> columnTypes = GetColumnTypes(table\_name);

try

{

List<string> setClauses = new List<string>();

List<string> setValues = new List<string>();

foreach (var column in values)

{

setClauses.Add($"{column.Column}");

setValues.Add($"@{column.Column}");

}

string setClause\_str = string.Join(",", setClauses);

string setValues\_str = string.Join(",", setValues);

string query = $"INSERT INTO {table\_name} ({setClause\_str}) VALUES ({setValues\_str});";

using (var cmd = new NpgsqlCommand(query, connection))

{

foreach (var (Column, Value) in values)

{

cmd.Parameters.AddWithValue($"@{Column}", GetConvertedValues(columnTypes, Column, Value));

}

cmd.ExecuteNonQuery();

}

}

catch (Exception ex)

{

Console.WriteLine("Помилка при додаванні данних " + ex.Message);

return true;

}

try

{

connection.Close();

}

catch (Exception ex)

{

Console.WriteLine("Помилка при закритті з'єднання з базою данних");

return true;

}

return false;

}

Дана функція вставляє нове значення до відповідної таблиці в БД, генеруючи для неї відповідний запрос, також приводить значення введені користувачем до відповідного типу.Для відслітковування помилок використовується try-catch.Функція оновлення даних:

public bool UpdateDataInTable(List<(string Column, string Value)> values\_res, string table\_name, int pk)

{

string pk\_str\_column = GetPrimaryKeyColumn(table\_name);

try

{

connection.Open();

}

catch (Exception ex)

{

Console.WriteLine("Помилка при підключенні до бази данних");

return true;

}

Dictionary<string, string> columnTypes = GetColumnTypes(table\_name);

try

{

List<string> setClauses = new List<string>();

foreach (var column in values\_res) {

setClauses.Add($"{column.Column} = @{column.Column}");

}

string setClause\_str = string.Join(",", setClauses);

string query = $"UPDATE {table\_name} SET {setClause\_str} WHERE {pk\_str\_column} = {pk};";

using (var cmd = new NpgsqlCommand(query, connection))

{

foreach (var (Column, Value) in values\_res)

{

cmd.Parameters.AddWithValue($"@{Column}", GetConvertedValues(columnTypes, Column, Value));

}

cmd.ExecuteNonQuery();

}

}

catch (Exception ex)

{

Console.WriteLine("Помилка при зміненні данних " + ex.Message);

return true;

}

try

{

connection.Close();

}

catch (Exception ex)

{

Console.WriteLine("Помилка при закритті з'єднання з базою данних");

return true;

}

return false;

}

Дана функція знаходить назву колонки де знаходиться pk таблиці для генерування строки запросу до відповідної таблиці в БД, після чого приводить отриманні дані від користувача і змінює відповідні дані.Для відслітковування помилок використовується try-catch.

Функція видалення значень:

public bool DeleteDataOfTable(string table\_name, int pk, string pk\_str)

{

try

{

connection.Open();

}

catch (Exception ex)

{

Console.WriteLine("Помилка при підключенні до бази данних");

return (true);

}

try

{

string query = $"DELETE FROM {table\_name} WHERE {pk\_str} = {pk}";

using (var cmd = new NpgsqlCommand(query, connection))

{

cmd.ExecuteNonQuery();

}

}

catch (Exception ex)

{

Console.WriteLine("Помилка при видаленні рядка");

return (true);

}

try

{

connection.Close();

}

catch (Exception ex)

{

Console.WriteLine("Помилка при закритті з'єднання з базою данних");

return (true);

}

return false;

}

Дана функція видаляє відповідний запис з обраної таблиці за pk, також у БД налаштоване CASCADE видалення, за рахунок цього будуть видалені відповідні записи в дочірніх таблицях.Для відслітковування помилок використовується try-catch.

Функція генерування даних:

public bool GenerateDataToCurrentTable(string proc\_name, int count\_rows)

{

try

{

connection.Open();

}

catch (Exception ex)

{

Console.WriteLine("Помилка при підключенні до бази данних");

return true;

}

try

{

string query = $"CALL {proc\_name}({count\_rows})";

using (var cmd = new NpgsqlCommand(query, connection))

{

cmd.CommandTimeout = 0;

cmd.ExecuteNonQuery();

}

}

catch (Exception ex)

{

Console.WriteLine("Помилка при генерації випадкових данних" + ex.Message);

return true;

}

try

{

connection.Close();

}

catch (Exception ex)

{

Console.WriteLine("Помилка при закритті з'єднання з базою данних");

return true;

}

return false;

}

Дана функція викликає відповідну процедуру з БД для генерування рандомних записів для відповідної таблиці.Для відслітковування помилок використовується try-catch.

**Приклади виконання вище наведених функцій**