



НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»  
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ  
**Кафедра системного програмування та спеціалізованих комп'ютерних  
систем**

**Лабораторна робота №2**  
з дисципліни  
«Бази даних і засоби управління»

Виконав студент III курсу  
ФПМ групи КВ-04  
Отрощенко А.В.  
Перевірив:

Київ – 2022

## Лабораторна робота № 2

### 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 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 2-х 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).

*Деталізоване завдання:*

1. Забезпечити можливість введення/редагування/вилучення даних у таблицях бази даних з можливістю контролю відповідності типів даних атрибутів таблиць (рядків, чисел, дати/часу). Для контролю пропонується два варіанти:

- а) контроль при введенні - валідація даних;
- б) перехоплення помилок (**try...except**) від сервера PostgreSQL при виконанні відповідної команди SQL.

Особливу увагу варто звернути на дані таблиць, що мають зв'язок 1:N.

З боку батьківської таблиці необхідно контролювати **вилучення (ON DELETE)** рядків за умови наявності даних у підлеглий таблиці.

З боку підлеглої таблиці варто контролювати наявність відповідного рядка у батьківській таблиці при виконанні **внесення** до неї нових даних.

Унеможливити виведення програмою на екрані системних помилок PostgreSQL шляхом їх перехоплення і адекватної обробки.

Внесення даних виконується користувачем у консольному вікні програми.

2. Забезпечити можливість автоматичної генерації великої кількості даних у таблицях за допомогою вбудованих у PostgreSQL функцій роботи з псевдовипадковими числами. Дані мають бути згенерованими **не програмою, а відповідним SQL-запитом!**

Приклад генерації 100 псевдовипадкових чисел:

```
select trunc(random()*1000)::int
from generate_series(1,100)
```

	trunc integer	
1	368	
2	773	
3	29	
4	66	
5	497	
6	956	

Приклад генерації 5-ти псевдовипадкових рядків:

```
select chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int)
from generate_series(1,5)
```

	?column? text	
1	NE	
2	MQ	
3	RN	
4	DW	
5	DA	

Приклад генерації псевдовипадкової мітки часу з діапазону [доступний за посиланням](#).

Кількість даних для генерування має вводити користувач з клавіатури.

Особливу увагу слід звернути на відповідність даних вимогам зовнішніх ключів з метою уникнення помилок порушення обмежень цілісності (foreign key).

3. Для реалізації багатокритеріального пошуку необхідно підготувати 3 запити, що включають дані з декількох таблиць і фільтрують рядки за 3-4 атрибутами цих таблиць. Забезпечити можливість введення конкретних значень констант для фільтрації з клавіатури користувачем. Після виведення даних вивести час виконання запиту у мілісекундах. Перевірити швидкодію роботи запитів на попередньо згенерованих даних.

4. Програмний код організувати згідно шаблону Model-View-Controller (MVC). Приклад організації коду згідно шаблону доступний [за даним посиланням](#). Модель, подання (представлення) та контролер мають бути реалізовані у окремих файлах. Для доступу до бази даних використовувати **лише мову SQL** (без ORM).

Рекомендована бібліотека взаємодії з PostgreSQL Psycopg2: <http://initd.org/psycopg/docs/usage.html>)

#### *Вимоги до інтерфейсу користувача*

Використовувати консольний інтерфейс користувача.

#### *Вимоги до інструментарію*

Середовище для лагодження SQL-запитів до бази даних – PgAdmin4.

Мова програмування – Python 3.6-3.7

Середовище розробки програмного забезпечення – PyCharm Community Edition 2020.

#### *Вимоги до оформлення звіту лабораторної роботи у електронному вигляді*

Опис (файл README.md) лабораторної роботи у **репозиторії GitHub** включає: назву лабораторної роботи, структуру бази даних з лабораторної роботи №1.

Репозиторій має містити файл звіту у форматі PDF та програмний код файлів мовою Python (або іншою).

Звіт у форматі PDF має містити: титульний аркуш, завдання та відповіді на вимоги до звітування щодо пунктів 1-4 деталізованого завдання:

#### *Вимоги до пункту №1 деталізованого завдання:*

- ілюстрації обробки виняткових ситуацій (помилки) при введенні/вилучення даних;

- ілюстрації валідації даних при введенні користувачем.

*Вимоги до пункту №2 деталізованого завдання:*

- копії екрану (ілюстрації) з фрагментами згенерованих даних таблиць.

*Вимоги до пункту №3 деталізованого завдання:*

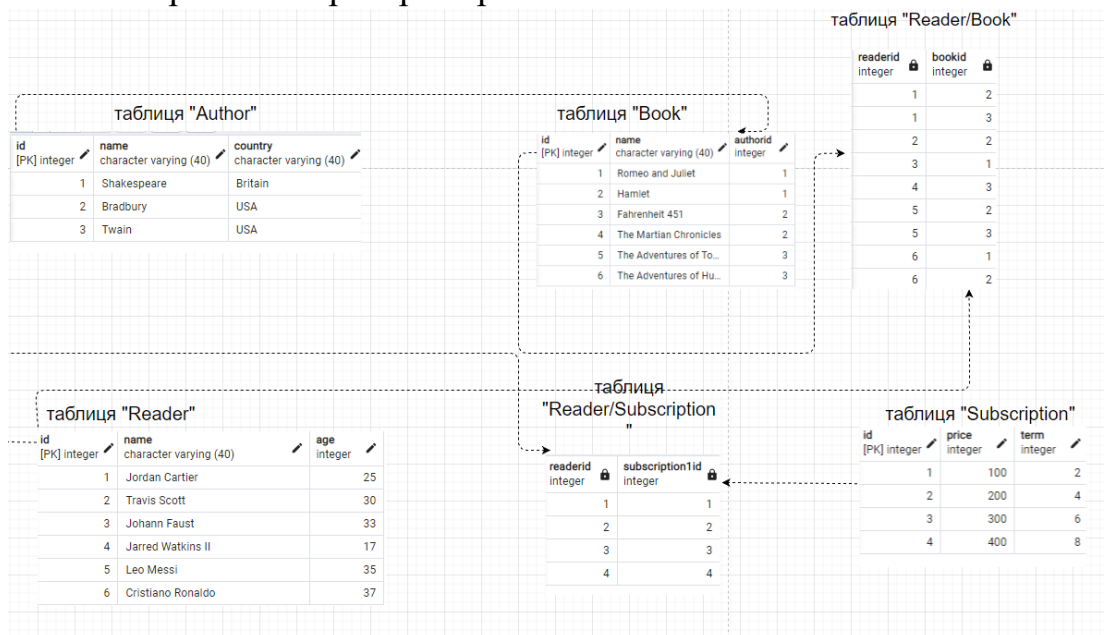
- ілюстрації введення пошукового запиту та результатів виконання запитів.

*Вимоги до пункту №4 деталізованого завдання:*

- ілюстрації програмного коду з репозиторію Git

## Нормалізована модель даних БД «Бібліотека»

На рисунку зображена нормалізована модель даних БД 'Бібліотека', розроблена на першій лабораторній роботі.



## Опис програми

Програма створена для управління базою даних за допомогою базових операцій СУБД PostgreSQL і мови програмування с#, та реалізовує функціональні вимоги, що наведені у завданні. Додаток використовує шаблон проектування MVC: model реалізує база даних, view – файл program.cs, controller – контролери для кожної сутності.

## Меню програми:

Головне меню програми з усіма таблицями

```
Choose table to operate with:
Press the number to choose a table
1.Author
2.Subscription
3.ReaderBook
4.Reader
5.ReaderSubscription
6.Book
0.Exit
```

Після вибору таблиці, відображається меню де можна вибрати яку операцію виконувати

```
What do you want to do with 'Author' table:
Press the number to choose an action
1.Create
2.Read
3.Update
4.Delete
5.Find
6.Generate
_
```

## Основні модулі програми

1. Program.cs – файл з меню;
2. BasedController.cs – файл з функціями для підключення та відключення БД від програми;
3. AuthorController.cs – контролер для таблиці author;
4. SubscriptionController.cs – контролер для таблиці subscription;
5. ReaderController.cs – контролер для таблиці reader;
6. ReaderSubscriptionController.cs – контролер для таблиці readersubscription;
7. ReaderBookController.cs – контролер для таблиці readerbook;
8. BookController.cs – контролер для таблиці book;

Щоб підключити БД до програми використовуємо мову програмування c# та бібліотеку NPGSQL

## Завдання 1

### Додавання даних до БД:

```
C:\Users\wifi0\source\repos\lab2_bd\bin\Debug
Provide the Author properties:
Name:
Dima
Country:
Ukraine_
```

### Результат:

```
C:\Users\wifi0\source\repos\lab2_bd\
Id: 25
Name: qOQR
Country: qUTR

Id: 26
Name: FYWR
Country: ZMUL

Id: 27
Name: bUCM
Country: qLEY

Id: 28
Name: rDGJ
Country: WLDK

Id: 29
Name: \IUA
Country: gDDE

Id: 30
Name: iTDG
Country: eTEW

Id: 31
Name: Dima
Country: Ukraine
```

### SQL-запит:

```
string sqlInsert = "Insert into author(name, country) VALUES(@name, @country)";
```

**Контроль наявності відповідного рядка у батьківській таблиці при виконанні внесення до дочірньої таблиці нових даних.**

Розглянемо на прикладі ReaderSubscription:

```
C:\Users\wifi0\source\repos\lab2_bd\bin\Debug\net6.0\lab2_bd.exe
Enter readersubscription properties:
Reader id:
312312
Subscription id:
2
23503: INSERT или UPDATE в таблице "readersubscription" нарушает ограничение внешнего ключа "reader_fk"
```

**Редагування даних:**

```
C:\Users\wifi0\source\repos\lab2_bd\bin\Debug\net6.0\lab2_bd.exe
Provide the name of field you want to find:
id
Provide the value in this field you want to find:
21
Provide the name of field you want to change:
name
Provide the new value in this field
Anton
```

**Результат:**

```
Id: 21
Name: Anton
Country: Ukraine
```

**SQL-запит:**

```
public readonly string sqlUpdate = "Update @table set @field_to_update = @new_value where @field_to_find = @old_value";
```

**Видалення даних:**

```
C:\Users\wifi0\source\repos\lab2_bd\bin\Debug\net6.0\lab2_bd.exe
What do you want to do with 'Author' table:
Press the number to choose an action
1.Create
2.Read
3.Update
4.Delete
5.Find
6.Generate
4
Provide the number of record you want to remove(0 to go back)
21_
```



Результат:

```
Id: 18  
Name: ^PDM  
Country: cGWL  
  
Id: 19  
Name: YHAG  
Country: BPDF  
  
Id: 20  
Name: OFFQ  
Country: DQOI
```

**Дослідження каскадного вилучення даних для таблиці ReaderSubscription**

```
CREATE TABLE IF NOT EXISTS public.readersubscription  
(  
    id integer NOT NULL DEFAULT nextval('readersubscription_id_seq'::regclass),  
    readerid integer NOT NULL,  
    subscriptionid integer NOT NULL,  
    CONSTRAINT reader_fk FOREIGN KEY (readerid)  
        REFERENCES public.reader (id) MATCH SIMPLE  
        ON UPDATE CASCADE  
        ON DELETE CASCADE,  
    CONSTRAINT subscription_fk FOREIGN KEY (subscriptionid)  
        REFERENCES public.subscription (id) MATCH SIMPLE  
        ON UPDATE CASCADE  
        ON DELETE CASCADE  
)
```

Візьмемо такий запис ReaderSubscription:

```
Id: 16  
Price: 537  
Term: 902
```

Видалимо запис з Subscription з Id = 16

```
C:\Users\wifi0\source\repos\lab2_bd\bin\Debug\net6.0\lab2_bd.exe
What do you want to do with 'Subscription' table:
Press the number to choose an action
1.Create
2.Read
3.Update
4.Delete
5.Find
6.Generate
4
Provide the number of record you want to remove(0 to go back)
16_
```

Відповідно запис, який ми взяли для прикладу з ReaderSubscription повинен видалитися.

```
C:\Users\wifi0\source\repos\lab2_bd\bin\Debug\net6.0\lab2_bd.exe
Id: 2
Reader Id: 2
Subscription Id: 2

Id: 5
Reader Id: 1
Subscription Id: 2

Id: 7
Reader Id: 2
Subscription Id: 2

Id: 8
Reader Id: 10
Subscription Id: 11

Id: 9
Reader Id: 11
Subscription Id: 12

Id: 10
Reader Id: 1
Subscription Id: 11
```

Як видно на зображенні, запис відсутній.

## Завдання 2. Пакетне генерування даних в таблиці Companies:

```
C:\Users\wifi0\source\repos\lab2_bd\bin\Debug\net6.0\lab2_bd.exe
What do you want to do with 'Author' table:
Press the number to choose an action
1.Create
2.Read
3.Update
4.Delete
5.Find
6.Generate
6
What amount of records do you want?
50000
```

### Результат

```
Id: 50016
Name: XPGW
Country: DSOE

Id: 50017
Name: iCEO
Country: ZIGJ

Id: 50018
Name: hXGX
Country: QFHI

Id: 50019
Name: mMAK
Country: XHQC

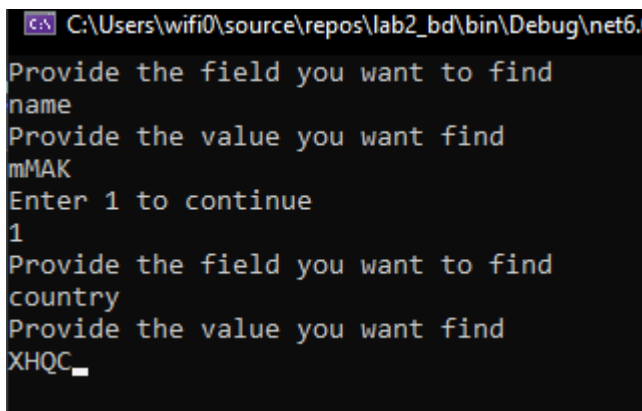
Id: 50020
Name: VYNM
Country: `LJK

Id: 50021
Name: YBEY
Country: LXNQ
```

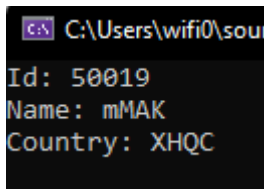
SQL-запит:

```
"insert into author(name, country)
(select chr(trunc(65 + random() * 50)::int) || chr(trunc(65 + random() * 25)::int) ||
chr(trunc(65 + random() * 25)::int) || chr(trunc(65 + random() * 25)::int),
chr(trunc(65 + random() * 50)::int) || chr(trunc(65 + random() * 25)::int) ||
chr(trunc(65 + random() * 25)::int) || chr(trunc(65 + random() * 25)::int)
from generate_series(1, 1000000) limit(50000))"
```

### Завдання 3. Пошук за двома-трьома атрибутами одночасно:



```
C:\Users\wifi0\source\repos\lab2_bd\bin\Debug\net6.
Provide the field you want to find
name
Provide the value you want find
mMAK
Enter 1 to continue
1
Provide the field you want to find
country
Provide the value you want find
XHQC
```



```
C:\Users\wifi0\source\repos\lab2_bd\bin\Debug\net6.
Id: 50019
Name: mMAK
Country: XHQC
```

Відповідний sql-запит:

“select id, name, country from author where name = ‘mMAK’ and country =’XHQC’”

### Код програми

Program.cs

```
using BD2.Controllers;
using System;

namespace BD2
{
    class Program
    {
        static void Main(string[] args)
        {
        }
```

```
String connectionString =  
"Host=localhost;Username=postgres;Password=123;Database=library1";
```

```
int table = 0;  
int action = 0;  
do  
{  
    table = FirstMenu();  
    if (table == 0)  
    {  
        return;  
    }  
  
    BaseController controller = null;  
  
    switch (table)  
    {  
        case 1:  
            action = SecondMenu("Author");  
            controller = new AuthorController(connectionString);  
            break;  
        case 2:  
            action = SecondMenu("Subscription");  
            controller = new SubscriptionController(connectionString);  
            break;  
        case 3:  
            action = SecondMenu("ReaderBook");  
            controller = new ReaderBookController(connectionString);  
            break;  
        case 4:  
            action = SecondMenu("Reader");  
            controller = new ReaderController(connectionString);  
            break;  
        case 5:  
            action = SecondMenu("ReaderSubscription");  
            controller = new  
ReaderSubscriptionController(connectionString);  
            break;  
        case 6:  
            action = SecondMenu("Book");  
            controller = new BookController(connectionString);  
            break;  
    }  
  
    switch (action)  
    {  
        case 1:  
            controller.Create();  
            break;  
        case 2:  
            controller.Read();  
            break;  
        case 3:  
            controller.Update();  
            break;  
        case 4:  

```

```

        controller.Delete();
        break;
    case 5:
        controller.Find();
        break;
    case 6:
        controller.Generate();
        break;
    }

    } while (true);
}

public static int FirstMenu()
{
    var choice = 0;
    var correct = false;
    do
    {
        Console.Clear();
        Console.WriteLine("Choose table to operate with:");
        Console.WriteLine("Press the number to choose a table");
        Console.WriteLine("1.Author");
        Console.WriteLine("2.Subscription");
        Console.WriteLine("3.ReaderBook");
        Console.WriteLine("4.Reader");
        Console.WriteLine("5.ReaderSubscription");
        Console.WriteLine("6.Book");
        Console.WriteLine("0.Exit");
        correct = Int32.TryParse(Console.ReadLine(), out choice);
    } while (choice < 0 || choice > 6 || correct == false);

    return choice;
}

public static int SecondMenu(string tableToChange)
{
    var choice = 0;
    var correct = false;
    do
    {
        Console.Clear();
        Console.WriteLine("What do you want to do with '" + tableToChange +
            "' table:");

        Console.WriteLine("Press the number to choose an action");
        Console.WriteLine("1.Create");
        Console.WriteLine("2.Read");
        Console.WriteLine("3.Update");
        Console.WriteLine("4.Delete");
        Console.WriteLine("5.Find");
        Console.WriteLine("6.Generate");
        correct = Int32.TryParse(Console.ReadLine(), out choice);
    } while (choice < 0 || choice > 6 || correct == false);
}

```

```

        return choice;
    }
}

```

BasedController.cs

```

using Npgsql;
using System;
using System.Collections.Generic;
using System.Text;

namespace BD2.Controllers
{
    public abstract class BasedController
    {
        public string connectionString;
        protected NpgsqlConnection sqlConnection;

        string fieldToFind = null;
        string valueToFind = null;
        string fieldToSet = null;
        string valueToSet = null;
        string[] fieldsToFind = new string[10];
        string[] valuesToFind = new string[10];

        public readonly string sqlRandomDate = "timestamp '2014-01-10 20:00:00' +
random() * (timestamp '2014-01-20 20:00:00' - timestamp '2014-01-10 10:00:00')";
        public readonly string sqlRandomBoolean =
"trunc(random()*2)::int::boolean";

        public readonly string sqlUpdate = "Update @table set @field_to_update =
@new_value where @field_to_find = @old_value";

        public readonly string sqlRandomString = "chr(trunc(65 + random() *
50)::int) || chr(trunc(65 + random() * 25)::int) || chr(trunc(65 + random() * 25)::int)
|| chr(trunc(65 + random() * 25)::int)";
        public readonly string sqlRandomInteger = "trunc(random()*1000)::int";

        public BasedController(string connectionString)
        {
            this.connectionString = connectionString;
            this.sqlConnection = new NpgsqlConnection(connectionString);
        }

        public virtual void Create()
        {
            throw new NotImplementedException();
        }
        public void Read()
        {
            Read("");
        }
        public virtual void Update()
    }
}

```

```

{
    throw new NotImplementedException();
}
public virtual void Delete()
{
    throw new NotImplementedException();
}
public virtual void Find()
{
    Console.Clear();
    int actualSize = 0;
    for (int i = 0; i < 10; i++)
    {
        Console.WriteLine("Provide the field you want to find ");
        fieldsToFind[i] = Console.ReadLine();
        Console.WriteLine("Provide the value you want find");
        valuesToFind[i] = Console.ReadLine();
        Console.WriteLine("Enter 1 to continue");
        actualSize++;

        int choose = 0;
        bool correct = Int32.TryParse(Console.ReadLine(), out choose);
        if (correct == false || choose != 1)
        {
            break;
        }
    }

    string whereCondition = " where ";

    int parseInt;
    if (Int32.TryParse(valuesToFind[0], out parseInt) == false)
    {
        valuesToFind[0] = "'" + valuesToFind[0] + "'";
    }
    whereCondition += fieldsToFind[0] + " = " + valuesToFind[0];

    for (int i = 1; i < actualSize; i++)
    {
        if (Int32.TryParse(valuesToFind[i], out parseInt) == false)
        {
            valuesToFind[i] = "'" + valuesToFind[i] + "'";
        }
        whereCondition += " and " + fieldsToFind[i] + " = " +
valuesToFind[i];
    }

    Read(whereCondition);
}
virtual public void Generate()
{
    throw new NotImplementedException();
}

virtual public void Read(string whereCondition)
{

```



```

    }

    protected void Delete(string sqlDelete)
    {
        bool correct = false;
        int id = 0;
        do
        {
            Console.WriteLine("Provide the number of record you want to
remove(0 to go back)");
            correct = Int32.TryParse(Console.ReadLine(), out id);
            if (correct == false)
            {
                Console.WriteLine("Id must be a number");
                Console.ReadLine();
                continue;
            }
        } while (correct == false || id < 0);

        sqlConnection.Open();

        using var cmd = new NpgsqlCommand(sqlDelete + id, sqlConnection);

        try
        {
            cmd.Prepare();
            cmd.ExecuteNonQuery();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            Console.ReadLine();
        }
        finally
        {
            sqlConnection.Close();
        }
    }

    private void Update(string table, string field_to_update, string new_value,
string field_to_find, string old_value)
    {
        sqlConnection.Open();

        StringBuilder updateString = new StringBuilder("Update", 200);

        int new_int;
        if (!Int32.TryParse(new_value, out new_int))
        {
            new_value = "'" + new_value + "'";
        }
        if (!Int32.TryParse(old_value, out new_int))
        {
            old_value = "'" + old_value + "'";
        }
    }

```

```

    }

    updateString.AppendFormat(" {0} set {1} = {2} where {3} = {4}", table,
field_to_update, new_value, field_to_find, old_value);

```

```

        using var cmd = new NpgsqlCommand(updateString.ToString(),
sqlConnection);

```

```

    try
    {
        cmd.Prepare();
        cmd.ExecuteNonQuery();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Console.ReadLine();
    }
    finally
    {
        sqlConnection.Close();
    }
}

```

```

protected void Update(string sqlUpdate)
{

```

```

    Console.Clear();
    Console.WriteLine("Provide the name of field you want to find:");
    fieldToFind = Console.ReadLine();
    Console.WriteLine("Provide the value in this field you want to find:");
    valueToFind = Console.ReadLine();

```

```

    Console.WriteLine("Provide the name of field you want to change:");
    fieldToSet = Console.ReadLine();
    Console.WriteLine("Provide the new value in this field");
    valueToSet = Console.ReadLine();

```

```

    int ParseInt = 0;
    if (Int32.TryParse(valueToFind, out ParseInt) == false)
    {
        valueToFind = "'" + valueToFind + "'";
    }
    if (Int32.TryParse(valueToSet, out ParseInt) == false)
    {
        valueToSet = "'" + valueToSet + "'";
    }

```

```

        string sqlQuery = sqlUpdate + "set " + fieldToSet + " = " + valueToSet
+ " where " + fieldToFind + " = " + valueToFind;

```

```

        sqlConnection.Open();

        using var cmd = new NpgsqlCommand(sqlQuery, sqlConnection);

        try
        {
            cmd.Prepare();
            cmd.ExecuteNonQuery();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            Console.ReadLine();
        }
        finally
        {
            sqlConnection.Close();
        }
    }

    protected void Generate(string sqlGenerate)
    {

        sqlConnection.Open();

        using var cmd = new NpgsqlCommand(sqlGenerate, sqlConnection);

        try
        {
            cmd.Prepare();
            cmd.ExecuteNonQuery();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            Console.ReadLine();
        }
        finally
        {
            sqlConnection.Close();
        }
    }
}

```

## AuthorController.cs

```

using Npgsql;
using System;
using System.Collections.Generic;
using System.Text;

namespace BD2.Controllers
{
    public class AuthorController : BaseController

```

```

    {
        public AuthorController(string connectionString) : base(connectionString) {
    }

    public override void Read(string whereCondition)
    {
        Console.Clear();

        sqlConnection.Open();

        string sqlSelect = "select id, name, country from author";

        using var cmd = new NpgsqlCommand(sqlSelect + whereCondition,
sqlConnection);
        try
        {
            using NpgsqlDataReader rdr = cmd.ExecuteReader();

            while (rdr.Read())
            {
                Console.WriteLine("Id: {0}", rdr.GetValue(0));
                Console.WriteLine("Name: {0}", rdr.GetValue(1));
                Console.WriteLine("Country: {0}", rdr.GetValue(2));
                Console.WriteLine();
            }
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            Console.ReadLine();
        }
        finally
        {
            sqlConnection.Close();
        }

        Console.ReadLine();
    }

    public override void Create()
    {
        string sqlInsert = "Insert into author(name, country) VALUES(@name,
@country)";

        string name = null;
        string country = null;

        bool correct = false;
        do
        {
            Console.Clear();
            Console.WriteLine("Provide the Author properties:");
            Console.WriteLine("Name:");
            name = Console.ReadLine();
            if (name.Length > 40)
            {
                correct = false;
            }
        }
    }
}

```

```

        Console.WriteLine("Length of name should be bigger than 40.");
        Console.ReadLine();
        continue;
    }

    Console.WriteLine("Country:");
    country = Console.ReadLine();
    if (country.Length > 40)
    {
        correct = false;
        Console.WriteLine("Length of country should be bigger than
40.");

        Console.ReadLine();
        continue;
    }

    correct = true;
} while (correct == false);

sqlConnection.Open();

using var cmd = new NpgsqlCommand(sqlInsert, sqlConnection);
cmd.Parameters.AddWithValue("name", name);
cmd.Parameters.AddWithValue("country", country);
cmd.Prepare();

try
{
    cmd.ExecuteNonQuery();
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
    Console.ReadLine();
}
finally
{
    sqlConnection.Close();
}
}

public override void Delete()
{
    base.Delete("delete from author where id = ");
}

public override void Update()
{
    base.Update("Update author ");
}

public override void Find()
{
    base.Find();
}

public override void Generate()

```

```

    {
        Console.WriteLine("What amount of records do you want?");
        bool correct = false;
        int recordsAmount;

        correct = Int32.TryParse(Console.ReadLine(), out recordsAmount);

        string sqlGenerate = "insert into author(name, country) (select "
            + base.sqlRandomString
            + ", "
            + base.sqlRandomString
            + " from generate_series(1, 1000000) limit(" + recordsAmount +
            "))";

        base.Generate(sqlGenerate);
    }
}

```

## SubscriptionController.cs

```

using Npgsql;
using System;
using System.Collections.Generic;
using System.Text;

namespace BD2.Controllers
{
    public class SubscriptionController : BaseController
    {
        public SubscriptionController(string connectionString) :
        base(connectionString) { }

        public override void Read(string whereCondition)
        {
            Console.Clear();

            sqlConnection.Open();

            string sqlSelect = "select id, price, term from subscription";

            using var cmd = new NpgsqlCommand(sqlSelect + whereCondition,
            sqlConnection);

            try
            {
                using NpgsqlDataReader rdr = cmd.ExecuteReader();

                while (rdr.Read())
                {
                    Console.WriteLine("Id: {0}", rdr.GetValue(0));
                    Console.WriteLine("Price: {0}", rdr.GetValue(1));
                    Console.WriteLine("Term: {0}", rdr.GetValue(2));
                    Console.WriteLine();
                }
                Console.WriteLine();
            }
        }
    }
}

```

```

    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Console.ReadLine();
    }
    finally
    {
        sqlConnection.Close();
    }

    Console.ReadLine();
}

public override void Create()
{
    string sqlInsert = "Insert into subscription(price, term)
VALUES(@price, @term)";

    int price = 0;
    int term = 0;

    bool correct = false;
    do
    {
        Console.Clear();
        Console.WriteLine("Enter Subscription properties:");
        Console.WriteLine("Price:");
        correct = Int32.TryParse(Console.ReadLine(), out price);
        if (correct == false)
        {
            Console.WriteLine("Price must be a number!");
            Console.ReadLine();
        }

        Console.WriteLine("Term:");
        correct = Int32.TryParse(Console.ReadLine(), out term);
        if (correct == false)
        {
            Console.WriteLine("Term must be a number!");
            Console.ReadLine();
        }
        correct = true;
    } while (correct == false);

    sqlConnection.Open();

    using var cmd = new NpgsqlCommand(sqlInsert, sqlConnection);
    cmd.Parameters.AddWithValue("price", price);
    cmd.Parameters.AddWithValue("term", term);
    cmd.Prepare();

    try
    {

```

```

        cmd.ExecuteNonQuery();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Console.ReadLine();
    }
    finally
    {
        sqlConnection.Close();
    }
}

public override void Delete()
{
    base.Delete("delete from subscription where id = ");
}
public override void Update()
{
    base.Update("Update subscription ");
}

public override void Generate()
{
    Console.WriteLine("What amount of records do you want?");
    bool correct = false;
    int recordsAmount;

    correct = Int32.TryParse(Console.ReadLine(), out recordsAmount);

    string sqlGenerate = "insert into subscription(price, term) (select "
        + base.sqlRandomInteger
        + ", "
        + base.sqlRandomInteger
        + " from generate_series(1, 1000000) limit(" + recordsAmount +
        "));";

    base.Generate(sqlGenerate);
}
}
}

```

## ReaderController.cs

```

using Npgsql;
using System;
using System.Collections.Generic;
using System.Text;

namespace BD2.Controllers
{
    public class ReaderController : BaseController
    {
        public ReaderController(string connectionString) : base(connectionString) {

```



```

public override void Read(string whereCondition)
{
    Console.Clear();

    sqlConnection.Open();

    string sqlSelect = "select id, name, age from reader";

    using var cmd = new NpgsqlCommand(sqlSelect + whereCondition,
sqlConnection);
    try
    {
        using NpgsqlDataReader rdr = cmd.ExecuteReader();

        while (rdr.Read())
        {
            Console.WriteLine("Id: {0}", rdr.GetValue(0));
            Console.WriteLine("Name: {0}", rdr.GetValue(1));
            Console.WriteLine("Age: {0}", rdr.GetValue(2));
            Console.WriteLine();
        }
        Console.WriteLine();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Console.ReadLine();
    }
    finally
    {
        sqlConnection.Close();
    }

    Console.ReadLine();
}

public override void Create()
{
    string sqlInsert = "Insert into reader(name, age) VALUES(@name, @age)";

    string name = null;
    int age = 0;

    bool correct = false;
    do
    {
        Console.Clear();
        Console.WriteLine("Enter reader properties:");
        Console.WriteLine("Name:");
        name = Console.ReadLine();
        if (name.Length > 40)
        {
            correct = false;
            Console.WriteLine("Length of name should be bigger than 40.");
        }
    }
}

```

```

        Console.ReadLine();
        continue;
    }

    Console.WriteLine("Age:");
    correct = Int32.TryParse(Console.ReadLine(), out age);
    if (correct == false)
    {
        Console.WriteLine("Age must be a number!");
        Console.ReadLine();
    }
    correct = true;
} while (correct == false);

sqlConnection.Open();

using var cmd = new NpgsqlCommand(sqlInsert, sqlConnection);
cmd.Parameters.AddWithValue("name", name);
cmd.Parameters.AddWithValue("age", age);
cmd.Prepare();

try
{
    cmd.ExecuteNonQuery();
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
    Console.ReadLine();
}
finally
{
    sqlConnection.Close();
}
}

public override void Delete()
{
    base.Delete("delete from reader where id = ");
}

public override void Update()
{
    base.Update("Update reader ");
}

public override void Generate()
{
    Console.WriteLine("What amount of records do you want?");
    bool correct = false;
    int recordsAmount;

    correct = Int32.TryParse(Console.ReadLine(), out recordsAmount);

    string sqlGenerate = "insert into reader(name, age) (select "
        + base.sqlRandomString
        + ", "

```

```

        + base.sqlRandomInteger
        + " from generate_series(1, 1000000) limit(" + recordsAmount +
    "));";
    base.Generate(sqlGenerate);
}
}
}

```

## ReaderSubscription.cs

```

using Npgsql;
using System;
using System.Collections.Generic;
using System.Text;

namespace BD2.Controllers
{
    public class ReaderSubscriptionController : BaseController
    {
        public ReaderSubscriptionController(string connectionString) :
base(connectionString) { }
        public override void Read(string whereCondition)
        {
            Console.Clear();

            sqlConnection.Open();

            string sqlSelect = "select id, readerid, subscriptionid from
readersubscription";

            using var cmd = new NpgsqlCommand(sqlSelect + whereCondition,
sqlConnection);
            try
            {
                using NpgsqlDataReader rdr = cmd.ExecuteReader();

                while (rdr.Read())
                {
                    Console.WriteLine("Id: {0}", rdr.GetValue(0));
                    Console.WriteLine("Reader Id: {0}", rdr.GetValue(1));
                    Console.WriteLine("Subscription Id: {0}", rdr.GetValue(2));
                    Console.WriteLine();
                }
                Console.WriteLine();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
                Console.ReadLine();
            }
            finally
            {
                sqlConnection.Close();
            }
        }
    }
}

```

```

        Console.ReadLine();
    }
    public override void Create()
    {
        string sqlInsert = "Insert into readersubscription (readerid,
subscriptionid) VALUES(@readerid, @subscriptionid)";

        int reader_id = 0;
        int subscription_id = 0;

        bool correct = false;
        do
        {
            Console.Clear();
            Console.WriteLine("Enter readersubscription properties:");
            Console.WriteLine("Reader id:");
            correct = Int32.TryParse(Console.ReadLine(), out reader_id);
            if (correct == false)
            {
                Console.WriteLine("Reader id must be a number!");
                Console.ReadLine();
            }

            Console.WriteLine("Subscription id:");
            correct = Int32.TryParse(Console.ReadLine(), out subscription_id);
            if (correct == false)
            {
                Console.WriteLine("Subscription id must be a number!");
                Console.ReadLine();
            }

            correct = true;
        } while (correct == false);

        sqlConnection.Open();

        using var cmd = new NpgsqlCommand(sqlInsert, sqlConnection);
        cmd.Parameters.AddWithValue("readerid", reader_id);
        cmd.Parameters.AddWithValue("subscriptionid", subscription_id);
        cmd.Prepare();

        try
        {
            cmd.ExecuteNonQuery();
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            Console.ReadLine();
        }
        finally
        {
            sqlConnection.Close();
        }
    }
}

```

```

    }
}
public override void Delete()
{
    base.Delete("delete from readersubscription where id = ");
}
public override void Update()
{
    base.Update("Update readersubscription ");
}

public override void Generate()
{
    Console.WriteLine("What amount of records do you want?");
    bool correct = false;
    int recordsAmount;
    correct = Int32.TryParse(Console.ReadLine(), out recordsAmount);
    string subquery = "with pa as ( select reader.id from reader
where reader.id not in ( select reader.id from readersubscription
join reader on readersubscription.readerid = reader.id ) limit(1)), "
+ "va as ( select subscription.id from subscription where
subscription.id not in ( select subscription.id from readersubscription
join subscription on readersubscription.subscriptionid = subscriptionid )
limit(1))";

    string sqlGenerate = subquery + " insert into
readersubscription(readerid, subscriptionid) (select pa.id, va.id from pa, va limit(1))";

    for (int i = 0; i < recordsAmount; i++)
    {
        base.Generate(sqlGenerate);
    }
}
}
}

```

## ReaderBook.cs

```

using Npgsql;
using System;
using System.Collections.Generic;
using System.Text;

namespace BD2.Controllers
{
    public class ReaderBookController : BaseController
    {
        public ReaderBookController(string connectionString) :
base(connectionString) { }
        public override void Read(string whereCondition)
        {
            Console.Clear();

            sqlConnection.Open();

            string sqlSelect = "select id, readerid, bookid from readerbook";

```

```

        using var cmd = new NpgsqlCommand(sqlSelect + whereCondition,
sqlConnection);
    try
    {
        using NpgsqlDataReader rdr = cmd.ExecuteReader();

        while (rdr.Read())
        {
            Console.WriteLine("Id: {0}", rdr.GetValue(0));
            Console.WriteLine("Reader Id: {0}", rdr.GetValue(1));
            Console.WriteLine("Book Id: {0}", rdr.GetValue(2));
            Console.WriteLine();
        }
        Console.WriteLine();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Console.ReadLine();
    }
    finally
    {
        sqlConnection.Close();
    }

    Console.ReadLine();
}
public override void Create()
{
    string sqlInsert = "Insert into readerbook (readerid, bookid)
VALUES(@readerid, @bookid)";

    int reader_id = 0;
    int book_id = 0;

    bool correct = false;
    do
    {
        Console.Clear();
        Console.WriteLine("Enter readerbook properties:");
        Console.WriteLine("Reader id:");
        correct = Int32.TryParse(Console.ReadLine(), out reader_id);
        if (correct == false)
        {
            Console.WriteLine("Reader id must be a number!");
            Console.ReadLine();
        }

        Console.WriteLine("Book id:");
        correct = Int32.TryParse(Console.ReadLine(), out book_id);
        if (correct == false)
        {
            Console.WriteLine("Book id must be a number!");
            Console.ReadLine();
        }
    }
}

```

```

        }

        correct = true;
    } while (correct == false);

    sqlConnection.Open();

    using var cmd = new NpgsqlCommand(sqlInsert, sqlConnection);
    cmd.Parameters.AddWithValue("readertid", reader_id);
    cmd.Parameters.AddWithValue("bookid", book_id);
    cmd.Prepare();

    try
    {
        cmd.ExecuteNonQuery();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        Console.ReadLine();
    }
    finally
    {
        sqlConnection.Close();
    }
}

public override void Delete()
{
    base.Delete("delete from readerbook where id = ");
}

public override void Update()
{
    base.Update("Update readerbook ");
}

public override void Generate()
{
    Console.WriteLine("What amount of records do you want?");
    bool correct = false;
    int recordsAmount;

    correct = Int32.TryParse(Console.ReadLine(), out recordsAmount);

    string sqlGenerate = "insert into readerbook(readerid, bookid) (select
reader.id, book.id"
        + " from reader, book limit(" + recordsAmount + "))";
    base.Generate(sqlGenerate);
}
}
}

```

BookController.cs

```

using Npgsql;
using System;
using System.Collections.Generic;
using System.Text;

namespace BD2.Controllers
{
    public class BookController : BaseController
    {
        public BookController(string connectionString) : base(connectionString) { }

        public override void Read(string whereCondition)
        {
            Console.Clear();

            sqlConnection.Open();

            string sqlSelect = "select id, name, authorId from book";

            using var cmd = new NpgsqlCommand(sqlSelect + whereCondition,
sqlConnection);
            try
            {
                using NpgsqlDataReader rdr = cmd.ExecuteReader();

                while (rdr.Read())
                {
                    Console.WriteLine("Id: {0}", rdr.GetValue(0));
                    Console.WriteLine("Name: {0}", rdr.GetValue(1));
                    Console.WriteLine("Author Id: {0}", rdr.GetValue(2));
                    Console.WriteLine();
                }
                Console.WriteLine();
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
                Console.ReadLine();
            }
            finally
            {
                sqlConnection.Close();
            }

            Console.ReadLine();
        }

        public override void Create()
        {
            string sqlInsert = "Insert into book (name, authorid) VALUES(@name,
@authorid)";

            string name = null;
            int author_id = 0;

```



```

bool correct = false;
do
{
    Console.Clear();
    Console.WriteLine("Enter Book properties:");
    Console.WriteLine("Name:");
    name = Console.ReadLine();
    if (name.Length > 40)
    {
        correct = false;
        Console.WriteLine("Length of name should be bigger than 40.");
        Console.ReadLine();
        continue;
    }

    Console.WriteLine("Author id:");
    correct = Int32.TryParse(Console.ReadLine(), out author_id);
    if (correct == false)
    {
        Console.WriteLine("Author id must be a number!");
        Console.ReadLine();
    }

    correct = true;
} while (correct == false);

sqlConnection.Open();

using var cmd = new NpgsqlCommand(sqlInsert, sqlConnection);
cmd.Parameters.AddWithValue("name", name);
cmd.Parameters.AddWithValue("authorid", author_id);
cmd.Prepare();

try
{
    cmd.ExecuteNonQuery();
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
    Console.ReadLine();
}
finally
{
    sqlConnection.Close();
}
}
public override void Delete()
{
    base.Delete("delete from book where id = ");
}
public override void Update()
{
    base.Update("Update book ");
}

```

```

    }

    public override void Generate()
    {
        Console.WriteLine("What amount of records do you want?");
        bool correct = false;
        int recordsAmount;

        correct = Int32.TryParse(Console.ReadLine(), out recordsAmount);

        string sqlGenerate = "insert into book(name, authorid) (select "
            + base.sqlRandomString
            + ", author.id from generate_series(1, 1000000), author limit(" +
recordsAmount + "))";
        base.Generate(sqlGenerate);
    }
}

```