

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 ІII курсу

ФПМ групи КВ-83

Трофімцов Д.С.

Перевірив: Павловський В.І.

Київ – 2020

**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 уведення/редагування/вилучення даних у таблицях бази даних з можливістю контролю відповідності типів даних атрибутів таблиць (рядків, чисел, дати/часу). Для контролю пропонується два варіанти:
2. контроль при введенні - валідація даних;
3. 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 підлеглій таблиці.

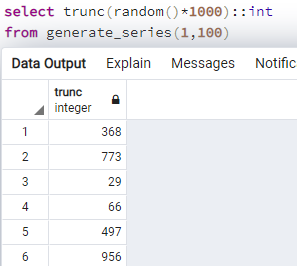
З боку підлеглої таблиці варто контролювати наявність відповідного рядка у батьківській таблиці при виконанні **внесення** до неї нових даних.

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

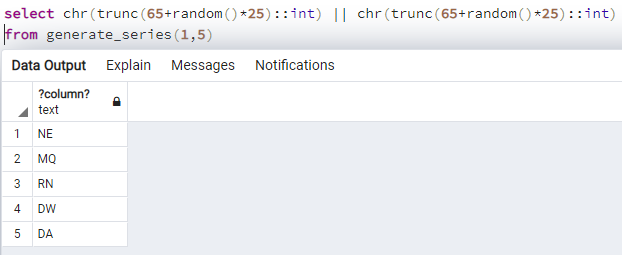
Внесення даних виконується користувачем у консольному вікні програми.

1. 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:



Приклад генерації 5-ти псевдовипадкових рядків:



Приклад генерації псевдовипадкової мітки часу з діапазону [доступний за посиланням](https://stackoverflow.com/questions/22964272/postgresql-get-a-random-datetime-timestamp-between-two-datetime-timestamp/22965061).

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).

1. 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 уведення 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.
2. Програмний код організувати згідно шаблону Model-View-Controller (MVC). Приклад організації коду згідно шаблону доступний [за даним посиланням](https://www.giacomodebidda.com/mvc-pattern-in-python-introduction-and-basicmodel/). 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 містить 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 (помилок) при уведенні/вилучення даних;
* ілюстрації валідації даних при уведенні користувачем.

*Вимоги до пункту №2 деталізованого завдання:*

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

*Вимоги до пункту №3 деталізованого завдання:*

* ілюстрації уведення пошукового запиту та результатів виконання запитів.

*Вимоги до пункту №4 деталізованого завдання:*

* ілюстрації програмного коду з репозиторію Git.

**Зміст**

[Нормалізована модель даних БД «Доставка їжі» 5](#_Toc57193385)

[Опис програми 6](#_Toc57193386)

[Меню програми: 6](#_Toc57193387)

[Основні модулі програми 6](#_Toc57193388)

[Завдання 1 7](#_Toc57193389)

[Додавання даних до БД: 7](#_Toc57193390)

[Редагування даних: 8](#_Toc57193391)

[Видалення даних: 9](#_Toc57193392)

[Дослідження каскадного вилучення даних для таблиці PatientCorona 10](#_Toc57193393)

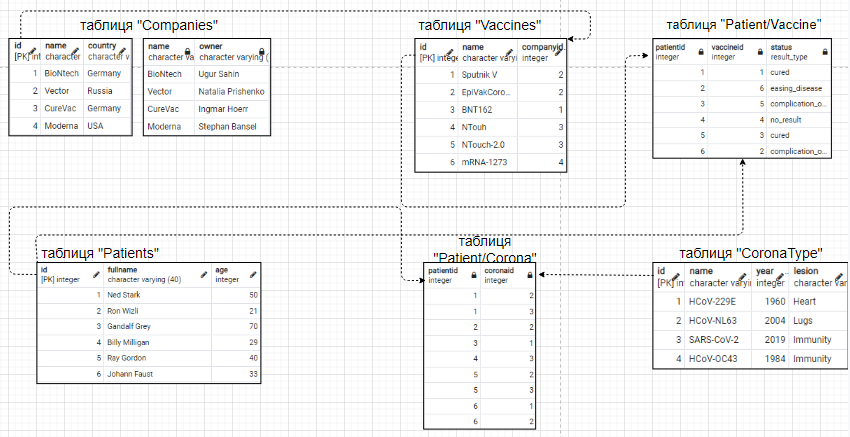
[Завдання 2. Пакетне генерування даних в таблиці Companies: 10](#_Toc57193394)

[Завдання 3. Пошук за двома-трьома атрибутами одночасно: 12](#_Toc57193395)

[Код програми 12](#_Toc57193396)

# Нормалізована модель даних БД «Тестування вакцини»

На рисунку зображена нормалізована модель даних БД ‘Тестування вакцини’, розроблена на першій лабораторній роботі.

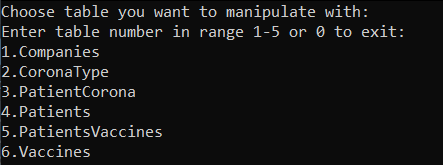


# Опис програми

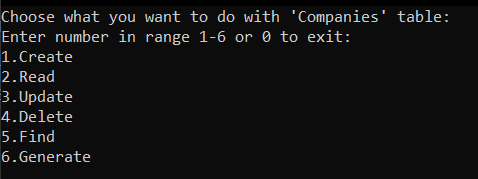
Програма створена для управління базою даних за допомогою базових операцій СУБД PostgreSQL і мови програмування **c#**, та реалізовує функціональні вимоги, що наведені у завданні. Додаток використовує шаблон проектування MVC: model реалізує база даних, view – файл program.cs, controller – контролери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

# Меню програми:

На зображенні показано головне меню програми з вибором таблиці, з якою хоче оперувати користувач.



Після вибору таблиці, відображається меню вибору операції.



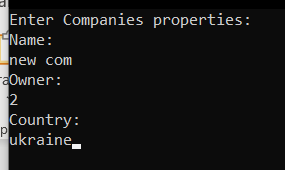
# Основні модулі програми

1. Program.cs – точка входу до програми, містить засоби обробки виключень та повідомлення помилок, викликає методи із контролерів;
2. BaseController.cs – містить функції та засоби для підключення (відключення) БД до (від) програми;
3. CompaniesController.cs – конкретний контролер для таблиці companies;
4. СoronaTypeController.cs – конкретний контролер для таблиці corona\_type;
5. PatientController.cs – конкретний контролер для таблиці patient;
6. PatientCoronaController.cs – конкретний контролер для таблиці patient\_corona;
7. PatientVaccineController.cs – конкретний контролер для таблиці patient\_vaccine;
8. VaccinesController.cs – конкретний контролер для таблиці vaccines;

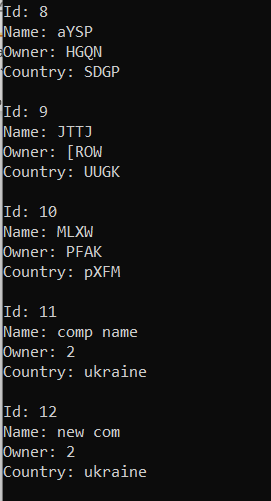
До БД програма підключається за допомогою бібліотеки NPGSQL для .NET.

# Завдання 1

## Додавання даних до БД:



Результат:

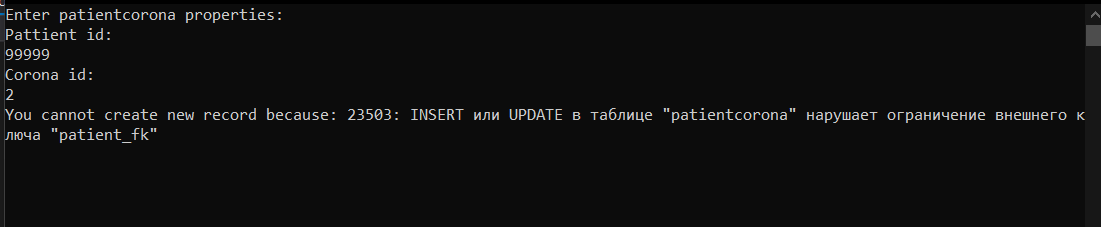


SQL-запит:

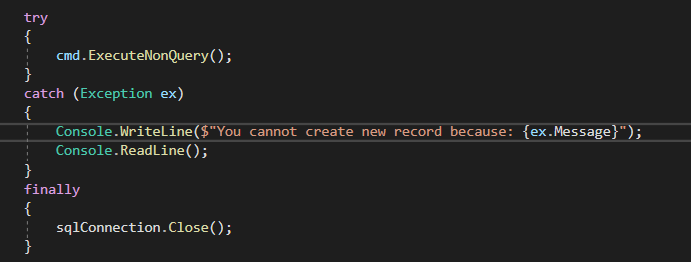


## Контроль наявності відповідного рядка у батьківській таблиці при виконанні внесення до дочірньої таблиці нових даних.

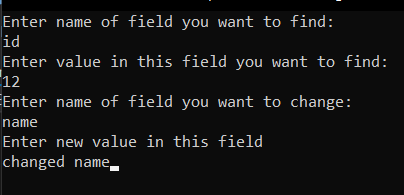
Розглянемо на прикладі PatientCorona:



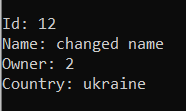
Код програми, який перехоплює помилку при створенні:



## Редагування даних:



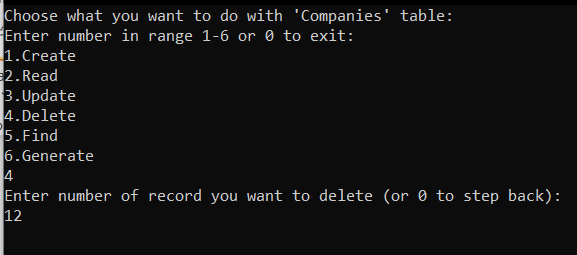
Результат:



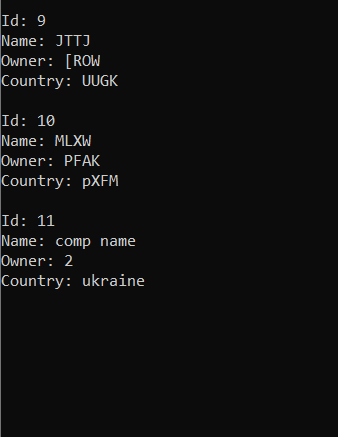
SQL-запит:



## Видалення даних:



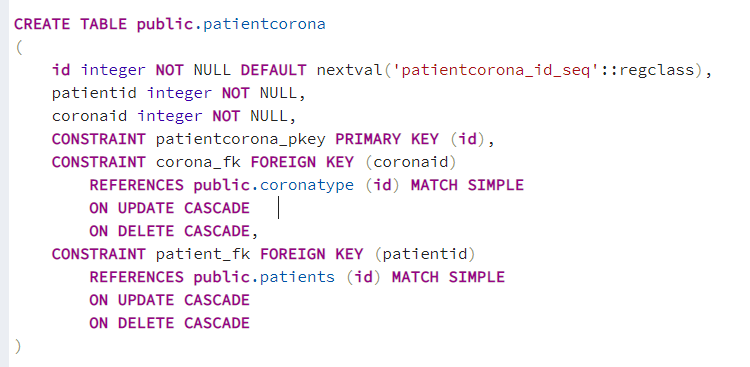
Результат:



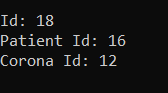
SQL-запит:



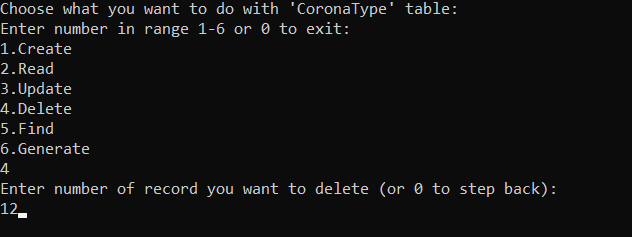
## Дослідження каскадного вилучення даних для таблиці PatientCorona



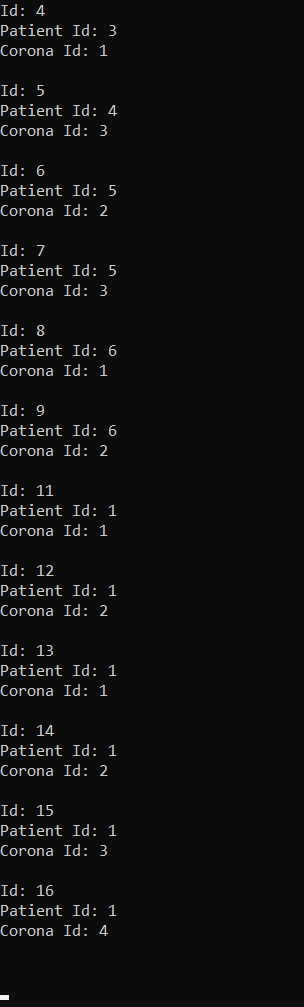
Візьмемо такий запис PatientCorona:



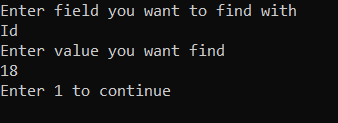
Видалимо запис з CoronaType з Id = 12



Відповідно запис, який ми взяли для прикладу з PatientCorona (з Id = 18) повинен видалитися. Подивимось всі записи:



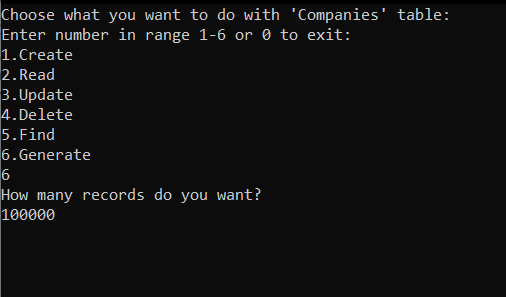
Як видно на зображенні, запис з Id = 18 відсутній. Спробуємо знайти його за допомогою пункту Find:



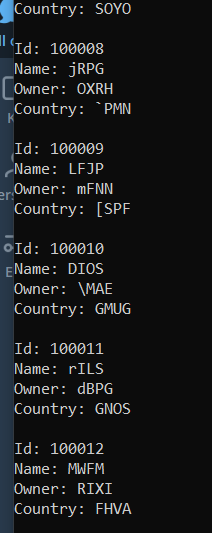
Результат (пустий екран, тобто запитів не знайдено):



# Завдання 2. Пакетне генерування даних в таблиці Companies:



Результат (всі запити не можуть поміститися, тому лише останні дані):



SQL-запит:

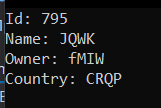
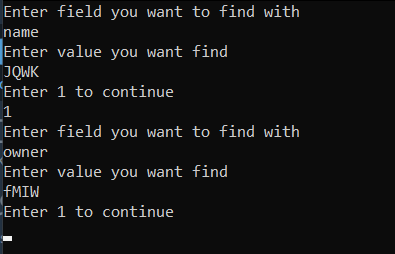
"insert into companies(name, owner, country)

(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),

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(100000))"

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



Відповідний sql-запит:



# Код програми

Лістинг модуля Program.cs

using BD2.Controllers;

using System;

namespace BD2

{

class Program

{

static void Main(string[] args)

{

String connectionString = "Host=localhost;Username=postgres;Password=p;Database=hospital";

int table = 0;

int action = 0;

do

{

table = FirstMenu();

if (table == 0)

{

return;

}

BaseController controller = null;

switch (table)

{

case 1:

action = SecondMenu("Companies");

controller = new CompaniesController(connectionString);

break;

case 2:

action = SecondMenu("CoronaType");

controller = new CoronaTypeController(connectionString);

break;

case 3:

action = SecondMenu("PatientCorona");

controller = new PatientCoronaController(connectionString);

break;

case 4:

action = SecondMenu("Patients");

controller = new PatientController(connectionString);

break;

case 5:

action = SecondMenu("PatientsVaccines");

controller = new PatientVaccineController(connectionString);

break;

case 6:

action = SecondMenu("Vaccines");

controller = new VaccinesController(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 you want to manipulate with:");

Console.WriteLine("Enter table number in range 1-5 or 0 to exit:");

Console.WriteLine("1.Companies");

Console.WriteLine("2.CoronaType");

Console.WriteLine("3.PatientCorona");

Console.WriteLine("4.Patients");

Console.WriteLine("5.PatientsVaccines");

Console.WriteLine("6.Vaccines");

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("Choose what you want to do with '" + tableToChange + "' table:");

Console.WriteLine("Enter number in range 1-6 or 0 to exit:");

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;

}

}

}

Лістинг модуля BaseController.cs

using Npgsql;

using System;

using System.Collections.Generic;

using System.Text;

namespace BD2.Controllers

{

public abstract class BaseController

{

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 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 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 BaseController(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("Enter field you want to find with");

fieldsToFind[i] = Console.ReadLine();

Console.WriteLine("Enter 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("Enter number of record you want to delete (or 0 to step 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("Enter name of field you want to find:");

fieldToFind = Console.ReadLine();

Console.WriteLine("Enter value in this field you want to find:");

valueToFind = Console.ReadLine();

Console.WriteLine("Enter name of field you want to change:");

fieldToSet = Console.ReadLine();

Console.WriteLine("Enter 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();

}

}

}

}

Лістинг модуля CompaniesController.cs

using Npgsql;

using System;

using System.Collections.Generic;

using System.Text;

namespace BD2.Controllers

{

public class CompaniesController : BaseController

{

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

public override void Read(string whereCondition)

{

Console.Clear();

sqlConnection.Open();

string sqlSelect = "select id, name, owner, country from companies";

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("Owner: {0}", rdr.GetValue(2));

Console.WriteLine("Country: {0}", rdr.GetValue(3));

Console.WriteLine();

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.ReadLine();

}

finally

{

sqlConnection.Close();

}

Console.ReadLine();

}

public override void Create()

{

string sqlInsert = "Insert into companies(name, owner, country) VALUES(@name, @owner, @country)";

string name = null;

string owner = null;

string country = null;

bool correct = false;

do

{

Console.Clear();

Console.WriteLine("Enter Companies properties:");

Console.WriteLine("Name:");

name = Console.ReadLine();

if (name.Length > 40)

{

correct = false;

Console.WriteLine("Length of name > 40. It is wrong.");

Console.ReadLine();

continue;

}

Console.WriteLine("Owner:");

owner = Console.ReadLine();

if (owner.Length > 40)

{

correct = false;

Console.WriteLine("Length of owner > 40. It is wrong.");

Console.ReadLine();

continue;

}

Console.WriteLine("Country:");

country = Console.ReadLine();

if (country.Length > 40)

{

correct = false;

Console.WriteLine("Length of country > 40. It is wrong.");

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("owner", owner);

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 companies where id = ");

}

public override void Update()

{

base.Update("Update companies ");

}

public override void Find()

{

base.Find();

}

public override void Generate()

{

Console.WriteLine("How many records do you want?");

bool correct = false;

int recordsAmount;

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

string sqlGenerate = "insert into companies(name, owner, country) (select "

+ base.sqlRandomString

+ ", "

+ base.sqlRandomString

+ ", "

+ base.sqlRandomString

+ " from generate\_series(1, 1000000) limit(" + recordsAmount + "))";

base.Generate(sqlGenerate);

}

}

}

Лістинг модуля CoronaTypeController.cs

using Npgsql;

using System;

using System.Collections.Generic;

using System.Text;

namespace BD2.Controllers

{

public class CoronaTypeController : BaseController

{

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

public override void Read(string whereCondition)

{

Console.Clear();

sqlConnection.Open();

string sqlSelect = "select id, name, year, lesion from coronatype";

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("Year: {0}", rdr.GetValue(2));

Console.WriteLine("Lesion: {0}", rdr.GetValue(3));

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 coronatype(name, year, lesion) VALUES(@name, @year, @lesion)";

string name = null;

int year = 0;

string lession = null;

bool correct = false;

do

{

Console.Clear();

Console.WriteLine("Enter CoronaType properties:");

Console.WriteLine("Name:");

name = Console.ReadLine();

if (name.Length > 40)

{

correct = false;

Console.WriteLine("Length of name > 40. It is wrong.");

Console.ReadLine();

continue;

}

Console.WriteLine("Year:");

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

if(correct == false)

{

Console.WriteLine("Year must be a number!");

Console.ReadLine();

}

Console.WriteLine("Lession:");

lession = Console.ReadLine();

correct = true;

} while (correct == false);

sqlConnection.Open();

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

cmd.Parameters.AddWithValue("name", name);

cmd.Parameters.AddWithValue("year", year);

cmd.Parameters.AddWithValue("lesion", lession);

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 coronatype where id = ");

}

public override void Update()

{

base.Update("Update coronatype ");

}

public override void Generate()

{

Console.WriteLine("How many records do you want?");

bool correct = false;

int recordsAmount;

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

string sqlGenerate = "insert into coronatype(name, year, lesion) (select "

+ base.sqlRandomString

+ ", "

+ base.sqlRandomInteger

+ ", "

+ base.sqlRandomString

+ " from generate\_series(1, 1000000) limit(" + recordsAmount + "))";

base.Generate(sqlGenerate);

}

}

}

Лістинг модуля PatientController.cs

using Npgsql;

using System;

using System.Collections.Generic;

using System.Text;

namespace BD2.Controllers

{

public class PatientController : BaseController

{

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

public override void Read(string whereCondition)

{

Console.Clear();

sqlConnection.Open();

string sqlSelect = "select id, fullname, age from patients";

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("Fullname: {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 patients(fullname, age) VALUES(@fullname, @age)";

string fullname = null;

int age = 0;

bool correct = false;

do

{

Console.Clear();

Console.WriteLine("Enter patient properties:");

Console.WriteLine("Fullname:");

fullname = Console.ReadLine();

if (fullname.Length > 40)

{

correct = false;

Console.WriteLine("Length of fullname > 40. It is wrong.");

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("fullname", fullname);

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 patients where id = ");

}

public override void Update()

{

base.Update("Update patients ");

}

public override void Generate()

{

Console.WriteLine("How many records do you want?");

bool correct = false;

int recordsAmount;

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

string sqlGenerate = "insert into patients(fullname, age) (select "

+ base.sqlRandomString

+ ", "

+ base.sqlRandomInteger

+ " from generate\_series(1, 1000000) limit(" + recordsAmount + "))";

base.Generate(sqlGenerate);

}

}

}

Лістинг модуля PatientCoronaController.cs

using Npgsql;

using System;

using System.Collections.Generic;

using System.Text;

namespace BD2.Controllers

{

public class PatientCoronaController : BaseController

{

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

public override void Read(string whereCondition)

{

Console.Clear();

sqlConnection.Open();

string sqlSelect = "select id, patientid, coronaid from patientcorona";

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("Patient Id: {0}", rdr.GetValue(1));

Console.WriteLine("Corona 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 patientcorona (patientid, coronaid) VALUES(@patientid, @coronaid)";

int pattient\_id = 0;

int corona\_id = 0;

bool correct = false;

do

{

Console.Clear();

Console.WriteLine("Enter patientcorona properties:");

Console.WriteLine("Pattient id:");

correct = Int32.TryParse(Console.ReadLine(), out pattient\_id);

if (correct == false)

{

Console.WriteLine("Pattient id must be a number!");

Console.ReadLine();

}

Console.WriteLine("Corona id:");

correct = Int32.TryParse(Console.ReadLine(), out corona\_id);

if (correct == false)

{

Console.WriteLine("Corona id must be a number!");

Console.ReadLine();

}

correct = true;

} while (correct == false);

sqlConnection.Open();

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

cmd.Parameters.AddWithValue("patientid", pattient\_id);

cmd.Parameters.AddWithValue("coronaid", corona\_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 patientcorona where id = ");

}

public override void Update()

{

base.Update("Update patientcorona ");

}

public override void Generate()

{

Console.WriteLine("How many records do you want?");

bool correct = false;

int recordsAmount;

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

string sqlGenerate = "insert into patientcorona(patientid, coronaid) (select patients.id, coronatype.id"

+ " from patients, coronatype limit(" + recordsAmount + "))";

base.Generate(sqlGenerate);

}

}

}

Лістинг модуля PatientVaccineController.cs

using Npgsql;

using System;

using System.Collections.Generic;

using System.Text;

namespace BD2.Controllers

{

public class PatientVaccineController : BaseController

{

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

public override void Read(string whereCondition)

{

Console.Clear();

sqlConnection.Open();

string sqlSelect = "select id, patientid, vaccineid from patientvaccine";

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("Patient Id: {0}", rdr.GetValue(1));

Console.WriteLine("Vaccine 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 patientvaccine (patientid, vaccineid) VALUES(@patientid, @vaccineid)";

int pattient\_id = 0;

int vaccine\_id = 0;

bool correct = false;

do

{

Console.Clear();

Console.WriteLine("Enter patientvaccine properties:");

Console.WriteLine("Pattient id:");

correct = Int32.TryParse(Console.ReadLine(), out pattient\_id);

if (correct == false)

{

Console.WriteLine("Pattient id must be a number!");

Console.ReadLine();

}

Console.WriteLine("Vaccine id:");

correct = Int32.TryParse(Console.ReadLine(), out vaccine\_id);

if (correct == false)

{

Console.WriteLine("Vaccine id must be a number!");

Console.ReadLine();

}

correct = true;

} while (correct == false);

sqlConnection.Open();

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

cmd.Parameters.AddWithValue("patientid", pattient\_id);

cmd.Parameters.AddWithValue("vaccineid", vaccine\_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 patientvaccine where id = ");

}

public override void Update()

{

base.Update("Update patientvaccine ");

}

public override void Generate()

{

Console.WriteLine("How many records do you want?");

bool correct = false;

int recordsAmount;

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

string subquery = "with pa as ( select patients.id from patients where patients.id not in ( select patients.id from patientvaccine join patients on patientvaccine.patientid = patients.id ) limit(1)), "

+ "va as ( select vaccines.id from vaccines where vaccines.id not in ( select vaccines.id from patientvaccine join vaccines on patientvaccine.vaccineid = vaccines.id ) limit(1))";

string sqlGenerate = subquery + " insert into patientvaccine(patientid, vaccineid) (select pa.id, va.id from pa, va limit(1))";

for (int i = 0; i < recordsAmount; i++)

{

base.Generate(sqlGenerate);

}

}

}

}

Лістинг модуля VaccinesController.cs

using Npgsql;

using System;

using System.Collections.Generic;

using System.Text;

namespace BD2.Controllers

{

public class VaccinesController : BaseController

{

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

public override void Read(string whereCondition)

{

Console.Clear();

sqlConnection.Open();

string sqlSelect = "select id, name, companyId from vaccines";

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("Company 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 vaccines (name, companyid) VALUES(@name, @companyid)";

string name = null;

int company\_id = 0;

bool correct = false;

do

{

Console.Clear();

Console.WriteLine("Enter Vaccine properties:");

Console.WriteLine("Name:");

name = Console.ReadLine();

if (name.Length > 40)

{

correct = false;

Console.WriteLine("Length of name > 40. It is wrong.");

Console.ReadLine();

continue;

}

Console.WriteLine("Company id:");

correct = Int32.TryParse(Console.ReadLine(), out company\_id);

if (correct == false)

{

Console.WriteLine("Company 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("companyid", company\_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 vaccines where id = ");

}

public override void Update()

{

base.Update("Update vaccines ");

}

public override void Generate()

{

Console.WriteLine("How many records do you want?");

bool correct = false;

int recordsAmount;

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

string sqlGenerate = "insert into vaccines(name, companyid) (select "

+ base.sqlRandomString

+ ", companies.id from generate\_series(1, 1000000), companies limit(" + recordsAmount + "))";

base.Generate(sqlGenerate);

}

}

}