

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

Левчук Є.

Перевірив: Павловський В.І.

Київ – 2020

**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.

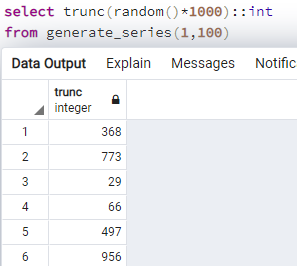
З боку підлеглої таблиці варто контролювати наявність відповідного рядка у батьківській таблиці при виконанні **внесення** до неї нових даних.

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

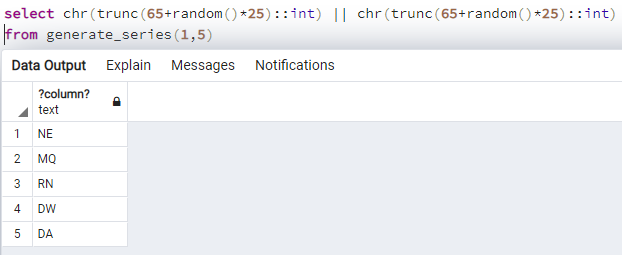
Внесення даних виконується користувачем у консольному вікні програми.

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 «Бібліотека» 5](#_Toc56774942)

[Результати роботи програми 7](#_Toc56774943)

[Завдання 1 7](#_Toc56774944)

[Додавання даних до БД: 7](#_Toc56774945)

[Редагування даних: 8](#_Toc56774946)

[Видалення даних: 9](#_Toc56774947)

[Контроль вилучення (ON DELETE) рядків батьківської таблиці за умови наявності даних у підлеглій таблиці. 9](#_Toc56774948)

[Ілюстрації обробки виняткових ситуацій (помилок) при уведенні/вилучення даних: 10](#_Toc56774949)

[Ілюстрації валідації даних при уведенні користувачем: 12](#_Toc56774950)

[Завдання 2. Пакетне генерування даних в таблиці Abonnement: 14](#_Toc56774951)

[Завдання 3. Пошук за двома-трьома атрибутами одночасно 15](#_Toc56774952)

[Завдання 4. Ілюстрація програмного коду з репозиторію GitHub: 17](#_Toc56774953)

[Program.cs: 17](#_Toc56774954)

[Папка Controllers: 19](#_Toc56774955)

[BaseController.cs: 20](#_Toc56774956)

[AbonnementController: 23](#_Toc56774957)

[AuthorController.cs: 25](#_Toc56774958)

[BookController.cs: 28](#_Toc56774959)

[GenreController.cs: 30](#_Toc56774960)

[ReaderCardController.cs: 32](#_Toc56774961)

[ReaderController.cs: 33](#_Toc56774962)

[Папка DAL: 36](#_Toc56774963)

[Папка Repositories: 36](#_Toc56774964)

[BaseRepository.cs: 37](#_Toc56774965)

[AbonnementRepository.cs: 38](#_Toc56774966)

[AuthorRepository.cs: 41](#_Toc56774967)

[BookRepository.cs: 43](#_Toc56774968)

[GenreRepository.cs: 46](#_Toc56774969)

[ReaderCardRepository.cs: 48](#_Toc56774970)

[ReaderRepository.cs: 50](#_Toc56774971)

[UnitOfWork.cs 52](#_Toc56774972)

[Папка Models 54](#_Toc56774973)

[BaseEntity.cs 54](#_Toc56774974)

[Abonnement.cs 54](#_Toc56774975)

[Author.cs 55](#_Toc56774976)

[Book.cs 55](#_Toc56774977)

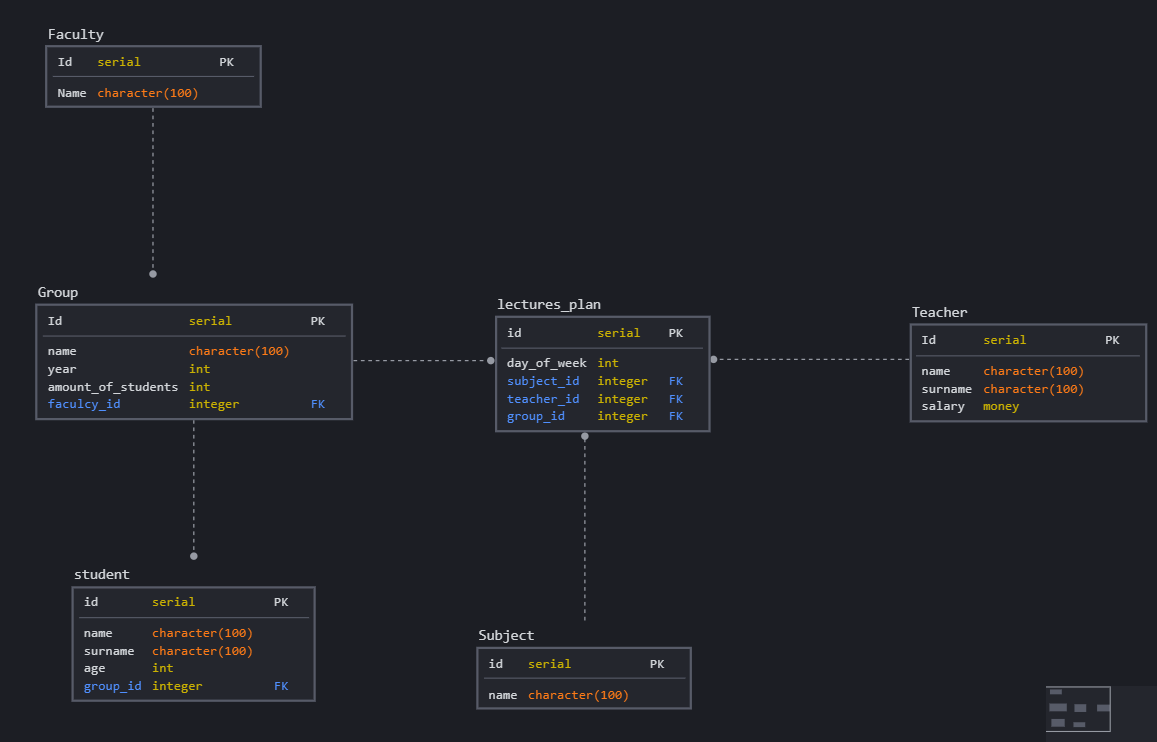
[Genre.cs 55](#_Toc56774978)

[Reader.cs 56](#_Toc56774979)

[ReaderCard.cs 56](#_Toc56774980)

# Нормалізована модель даних БД «Університет»

Програма використовує базу даних ‘Університет, розроблену на попередній лабораторній роботі. На рисунку зображена нормалізована модель бази даних.



Опис програми

Програма створена для управління базою даних за допомогою базових операцій СУБД PostgreSQL та реалізовує функціональні вимоги, що наведені у завданні. Програма умовно поділена на дві частини (містить дві папки):

1. Models – моделі таблиць з бази даних.
2. Controllers – контролери, що місять базові операції для кожної з таблиць.

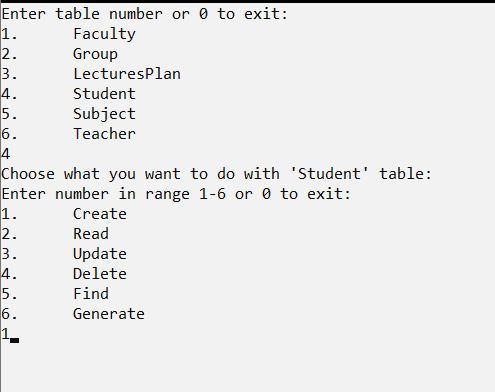
# Результати роботи програми

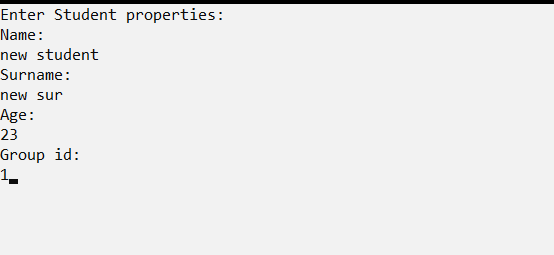
Запуск програми:

# Завдання 1

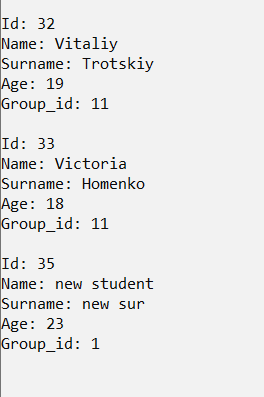
## 

## Додавання даних до БД:

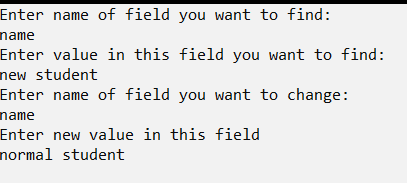




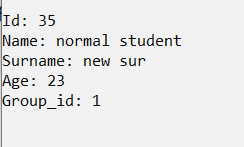
Результат:



## Редагування даних:



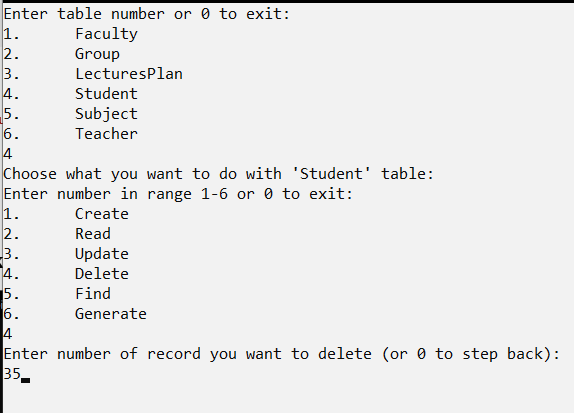
Результат:



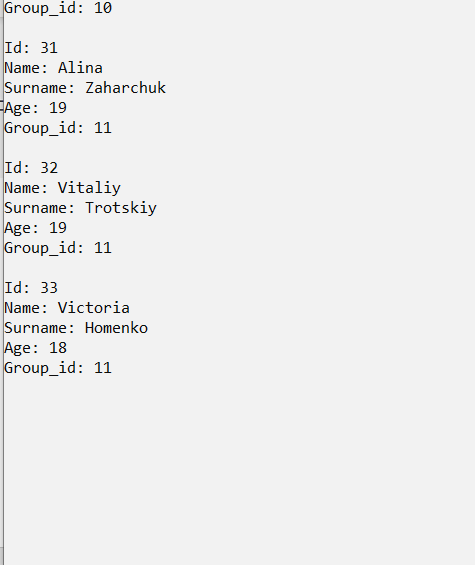
SQL-запит:

Insert into students(name, surname, age, group\_id) VALUES(@name, @surname, @age, @group\_id)

## Видалення даних:



Результат (запис відсутній):

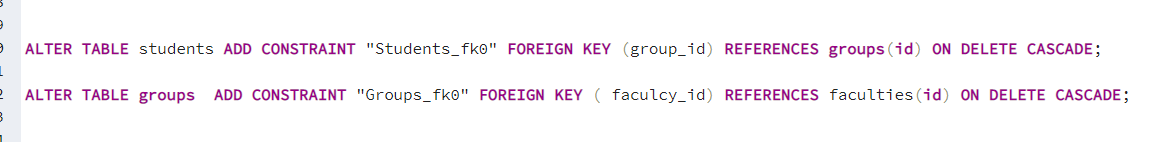


SQL-запит:

Delete from students where id = @id

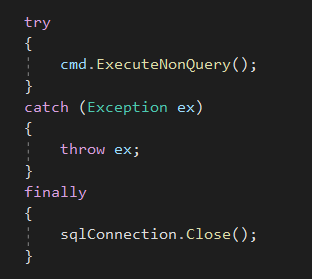
## Контроль вилучення (ON DELETE) рядків батьківської таблиці за умови наявності даних у підлеглій таблиці.

Як видно на зображенні, база даних передбачає видалення зв’язаних таблиць при видаленні ‘батьківської’.

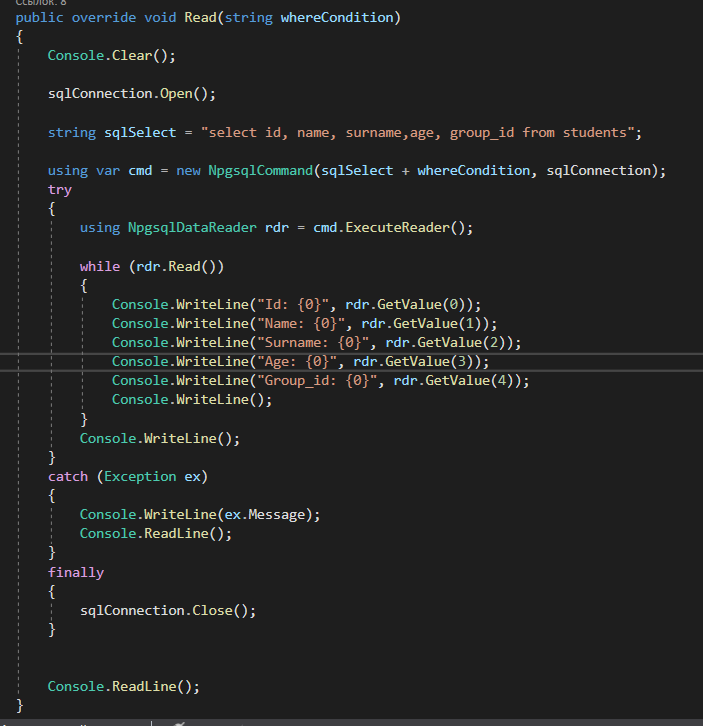


## Ілюстрації обробки виняткових ситуацій (помилок) при уведенні/вилучення даних:

Ілюстрація обробки виняткових ситуацій(помилок) при введенні/видалянні, генерації даних в коді.

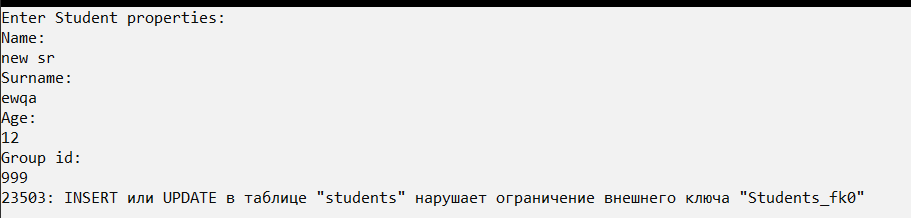


Ілюстрація обробки виняткових ситуацій при зчитуванні даних з таблиці.

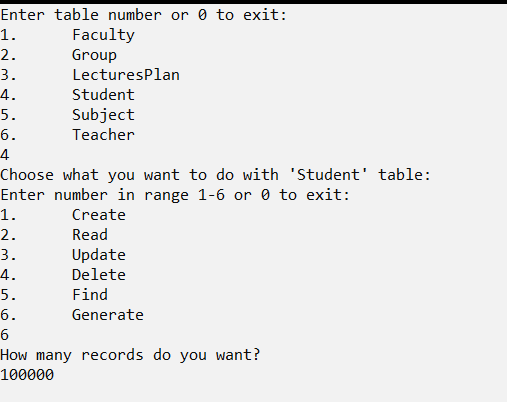


## Ілюстрації валідації даних при уведенні користувачем:

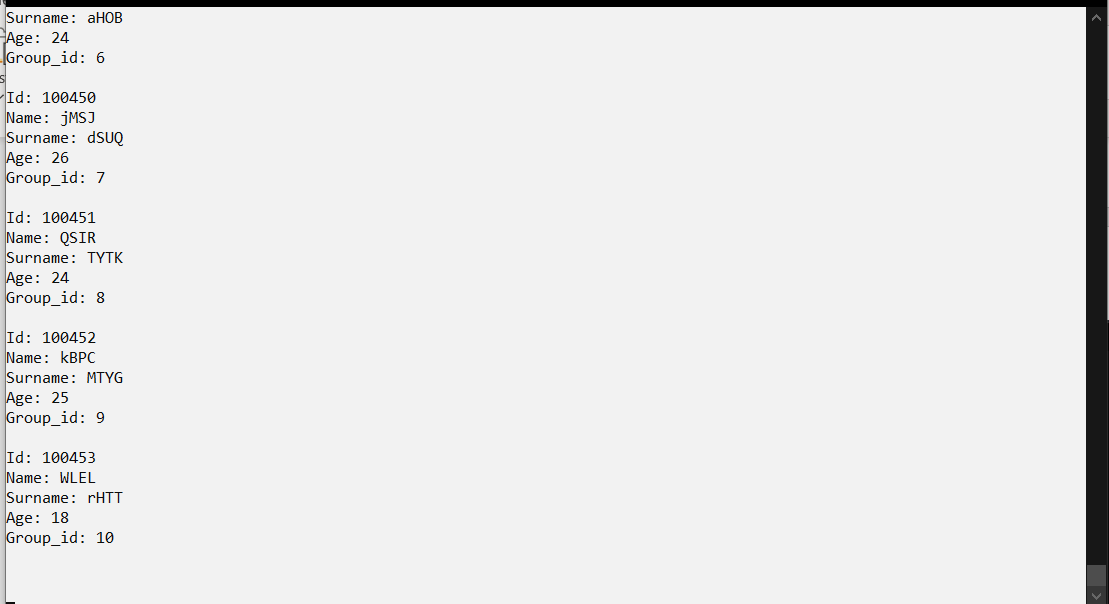
Якщо користувач вводить невірний зовнішній ключ, програма повідомляє про помилку (приклад – таблиця students).



# Завдання 2. Пакетне генерування даних в таблиці Abonnement:



Результат:



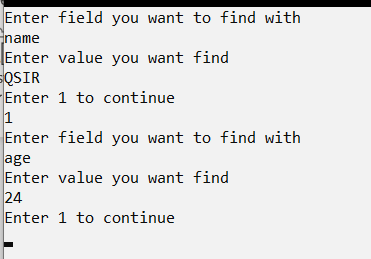
SQL-запит:

insert into students(name, surname, age, group\_id)

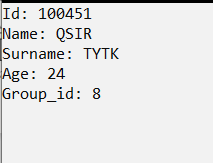
(select chr(trunc(65 + random() \* 50)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int), chr(trunc(65 + random() \* 50)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int) || chr(trunc(65 + random() \* 25)::int), 18 + trunc(random() \* 10)::int, groups.id

from generate\_series(1, 1000000), groups limit(100000))"

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



Отримали результат, в якому Name = `QSIR` AND age = 24.



**Відповідний sql-запит:**

Select \*

from students

where name = 'QSIR' and age = 24

# Завдання 4. Ілюстрація програмного коду з репозиторію

# Program.cs:

using BDLZ.Controllers;

using System;

namespace BDLZ

{

class Program

{

static void Main(string[] args)

{

Console.BackgroundColor = ConsoleColor.White;

Console.ForegroundColor = ConsoleColor.Black;

string connectionString = "Host=localhost;Username=postgres;Password=p;Database=bdlz2";

int table = 0;

int action = 0;

do

{

table = FirstMenu();

if (table == 0)

{

return;

}

BaseController controller = null;

switch (table)

{

case 1:

action = SecondMenu("Faculty");

controller = new FacultyController(connectionString);

break;

case 2:

action = SecondMenu("Group");

controller = new GroupController(connectionString);

break;

case 3:

action = SecondMenu("Lectures");

controller = new LecturesPlanController(connectionString);

break;

case 4:

action = SecondMenu("Student");

controller = new StudentController(connectionString);

break;

case 5:

action = SecondMenu("Subject");

controller = new SubjectController(connectionString);

break;

case 6:

action = SecondMenu("Teacher");

controller = new TeacherController(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("Enter table number or 0 to exit:");

Console.WriteLine("1.\tFaculty");

Console.WriteLine("2.\tGroup");

Console.WriteLine("3.\tLecturesPlan");

Console.WriteLine("4.\tStudent");

Console.WriteLine("5.\tSubject");

Console.WriteLine("6.\tTeacher");

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.\tCreate");

Console.WriteLine("2.\tRead");

Console.WriteLine("3.\tUpdate");

Console.WriteLine("4.\tDelete");

Console.WriteLine("5.\tFind");

Console.WriteLine("6.\tGenerate");

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

} while (choice < 0 || choice > 6 || correct == false);

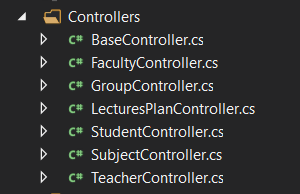
return choice;

}

}

}

## Папка Controllers:



## BaseController.cs:

using Npgsql;

using System;

using System.Collections.Generic;

using System.Text;

namespace BDLZ.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();

}

}

}

}

FacultyController: using Npgsql;

using System;

using System.Collections.Generic;

using System.Text;

namespace BDLZ.Controllers

{

public class FacultyController : BaseController

{

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

public override void Read(string whereCondition)

{

Console.Clear();

sqlConnection.Open();

string sqlSelect = "select id, name from faculties";

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();

}

Console.WriteLine();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.ReadLine();

}

finally

{

sqlConnection.Close();

}

Console.ReadLine();

}

public override void Create()

{

string sqlInsert = "Insert into faculties(name) VALUES(@name)";

string name = null;

Console.Clear();

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

Console.WriteLine("Name:");

name = Console.ReadLine();

sqlConnection.Open();

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

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

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 faculties where id = ");

}

public override void Update()

{

base.Update("Update faculties ");

}

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 faculty(name) (select "

+ base.sqlRandomString

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

base.Generate(sqlGenerate);

}

}

}

## GroupController.cs:

using BDLZ.Models;

using Npgsql;

using System;

using System.Collections.Generic;

using System.Text;

namespace BDLZ.Controllers

{

public class GroupController : BaseController

{

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

public override void Read(string whereCondition)

{

Console.Clear();

sqlConnection.Open();

string sqlSelect = "select id, name, year, amount\_of\_students, faculcy\_id from groups";

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("amount\_of\_students: {0}", rdr.GetValue(3));

Console.WriteLine("faculty\_id: {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 groups(name, year, amount\_of\_students, faculcy\_id) VALUES(@name, @year, @amount\_of\_students, @faculcy\_id)";

var group = new Group();

Console.Clear();

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

Console.WriteLine("Name:");

group.name = Console.ReadLine();

Console.WriteLine("Owner:");

group.year = Int32.Parse(Console.ReadLine());

Console.WriteLine("Amount of students:");

group.amount\_of\_students = Int32.Parse(Console.ReadLine());

Console.WriteLine("Faculty id:");

group.faculty\_id = Int32.Parse(Console.ReadLine());

sqlConnection.Open();

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

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

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

cmd.Parameters.AddWithValue("amount\_of\_students", group.amount\_of\_students);

cmd.Parameters.AddWithValue("faculcy\_id", group.faculty\_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 groups where id = ");

}

public override void Update()

{

base.Update("Update groups ");

}

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 groups(name, year, amount\_of\_students, faculcy\_id) (select "

+ base.sqlRandomString

+ ", "

+ base.sqlRandomInteger

+ ", "

+ base.sqlRandomInteger

+ ", "

+ base.sqlRandomInteger

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

base.Generate(sqlGenerate);

}

}

}

## LecturesPlanController.cs:

using BDLZ.Models;

using Npgsql;

using System;

using System.Collections.Generic;

using System.Text;

namespace BDLZ.Controllers

{

public class LecturesPlanController : BaseController

{

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

public override void Read(string whereCondition)

{

Console.Clear();

sqlConnection.Open();

string sqlSelect = "select id, group\_id, subject\_id, teacher\_id from lectures\_plan";

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("group\_id: {0}", rdr.GetValue(1));

Console.WriteLine("subject\_id: {0}", rdr.GetValue(2));

Console.WriteLine("teacher\_id: {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 lectures\_plan(group\_id, subject\_id, teacher\_id) VALUES(@group\_id, @subject\_id, @teacher\_id)";

var lecturesPlan = new LecturesPlan();

Console.Clear();

Console.WriteLine("Enter Lectures\_plan properties:");

Console.WriteLine("Group id:");

lecturesPlan.group\_id = Int32.Parse(Console.ReadLine());

Console.WriteLine("Subject id:");

lecturesPlan.subject\_id = Int32.Parse(Console.ReadLine());

Console.WriteLine("Teacher id:");

lecturesPlan.teacher\_id = Int32.Parse(Console.ReadLine());

sqlConnection.Open();

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

cmd.Parameters.AddWithValue("group\_id", lecturesPlan.group\_id);

cmd.Parameters.AddWithValue("subject\_id", lecturesPlan.subject\_id);

cmd.Parameters.AddWithValue("teacher\_id", lecturesPlan.teacher\_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 lectures\_plan where id = ");

}

public override void Update()

{

base.Update("Update lectures\_plan ");

}

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 lectures\_plan(group\_id, subject\_id, teacher\_id) (select "

+ "groups.id, subjects.id, teachers.id from groups, subjects, teachers limit(" + recordsAmount + "))";

base.Generate(sqlGenerate);

}

}

}

## StudentController.cs:

using BDLZ.Models;

using Npgsql;

using System;

using System.Collections.Generic;

using System.Text;

namespace BDLZ.Controllers

{

public class StudentController : BaseController

{

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

public override void Read(string whereCondition)

{

Console.Clear();

sqlConnection.Open();

string sqlSelect = "select id, name, surname,age, group\_id from students";

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

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

Console.WriteLine("Group\_id: {0}", rdr.GetValue(4));

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 students(name, surname, age, group\_id) VALUES(@name, @surname, @age, @group\_id)";

var student = new Student();

Console.Clear();

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

Console.WriteLine("Name:");

student.name = Console.ReadLine();

Console.WriteLine("Surname:");

student.surname = Console.ReadLine();

Console.WriteLine("Age:");

student.age = Int32.Parse(Console.ReadLine());

Console.WriteLine("Group id:");

student.group\_id = Int32.Parse(Console.ReadLine());

sqlConnection.Open();

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

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

cmd.Parameters.AddWithValue("surname", student.surname);

cmd.Parameters.AddWithValue("group\_id", student.group\_id);

cmd.Parameters.AddWithValue("age", student.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 students where id = ");

}

public override void Update()

{

base.Update("Update students ");

}

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 students(name, surname, age, group\_id) (select "

+ base.sqlRandomString

+ ", "

+ base.sqlRandomString

+ ", 18 + trunc(random() \* 10)::int"

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

base.Generate(sqlGenerate);

}

}

}

## SubjectController.cs:

using BDLZ.Models;

using Npgsql;

using System;

using System.Collections.Generic;

using System.Text;

namespace BDLZ.Controllers

{

public class SubjectController : BaseController

{

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

public override void Read(string whereCondition)

{

Console.Clear();

sqlConnection.Open();

string sqlSelect = "select id, name from subjects";

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();

}

Console.WriteLine();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.ReadLine();

}

finally

{

sqlConnection.Close();

}

Console.ReadLine();

}

public override void Create()

{

string sqlInsert = "Insert into subjects(name) VALUES(@name)";

var subject = new Subject();

Console.Clear();

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

Console.WriteLine("Name:");

subject.name = Console.ReadLine();

sqlConnection.Open();

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

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

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 subjects where id = ");

}

public override void Update()

{

base.Update("Update subjects ");

}

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 subjects(name) (select "

+ base.sqlRandomString

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

base.Generate(sqlGenerate);

}

}

}

## TeacherController.cs:

using BDLZ.Models;

using Npgsql;

using System;

using System.Collections.Generic;

using System.Text;

namespace BDLZ.Controllers

{

public class TeacherController : BaseController

{

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

public override void Read(string whereCondition)

{

Console.Clear();

sqlConnection.Open();

string sqlSelect = "select id, name, surname, salary from groups";

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

Console.WriteLine("Salary: {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 teacher(name, surname, salary) VALUES(@name, @surname, @salary)";

var teacher = new Teacher();

Console.Clear();

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

Console.WriteLine("Name:");

teacher.name = Console.ReadLine();

Console.WriteLine("Surname:");

teacher.surname = Console.ReadLine();

Console.WriteLine("Salary:");

teacher.salary = Decimal.Parse(Console.ReadLine());

sqlConnection.Open();

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

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

cmd.Parameters.AddWithValue("surname", teacher.surname);

cmd.Parameters.AddWithValue("salary", teacher.salary);

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 teachers where id = ");

}

public override void Update()

{

base.Update("Update teachers ");

}

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 teachers(name, surname, salary) (select "

+ base.sqlRandomString

+ ", "

+ base.sqlRandomString

+ ", "

+ base.sqlRandomInteger

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

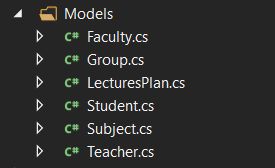
base.Generate(sqlGenerate);

}

}

}

## Папка Models:



## Faculty.cs:

using System;

using System.Collections.Generic;

using System.Text;

namespace BDLZ.Models

{

public class Faculty

{

public int id { get; set; }

public string name { get; set; }

}

}

## Group.cs:

using System;

using System.Collections.Generic;

using System.Text;

namespace BDLZ.Models

{

public class Group

{

public int id { get; set; }

public string name { get; set; }

public int year { get; set; }

public int amount\_of\_students { get; set; }

public int faculty\_id { get; set; }

}

}

## LecturesPlan.cs:

using System;

using System.Collections.Generic;

using System.Text;

namespace BDLZ.Models

{

public class LecturesPlan

{

public int id { get; set; }

public int day\_of\_week { get; set; }

public int group\_id { get; set; }

public int subject\_id { get; set; }

public int teacher\_id { get; set; }

}

}

## Student.cs:

using System;

using System.Collections.Generic;

using System.Text;

namespace BDLZ.Models

{

public class Student

{

public int id { get; set; }

public string name { get; set; }

public string surname { get; set; }

public int age { get; set; }

public int group\_id { get; set; }

}

}

## Subject.cs:

using System;

using System.Collections.Generic;

using System.Text;

namespace BDLZ.Models

{

public class Subject

{

public int id { get; set; }

public string name { get; set; }

}

}

## Teacher.cs:

using System;

using System.Collections.Generic;

using System.Text;

namespace BDLZ.Models

{

public class Teacher

{

public int id { get; set; }

public string name { get; set; }

public string surname { get; set; }

public decimal salary { get; set; }

}

}