

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

Непочатих 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.

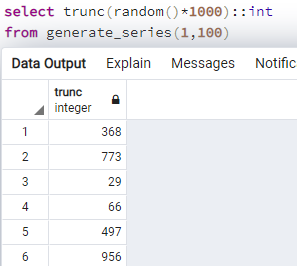
З боку підлеглої таблиці варто контролювати наявність відповідного рядка у батьківській таблиці при виконанні **внесення** до неї нових даних.

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

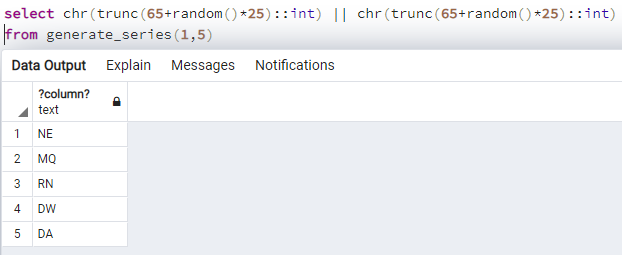
Внесення даних виконується користувачем у консольному вікні програми.

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)

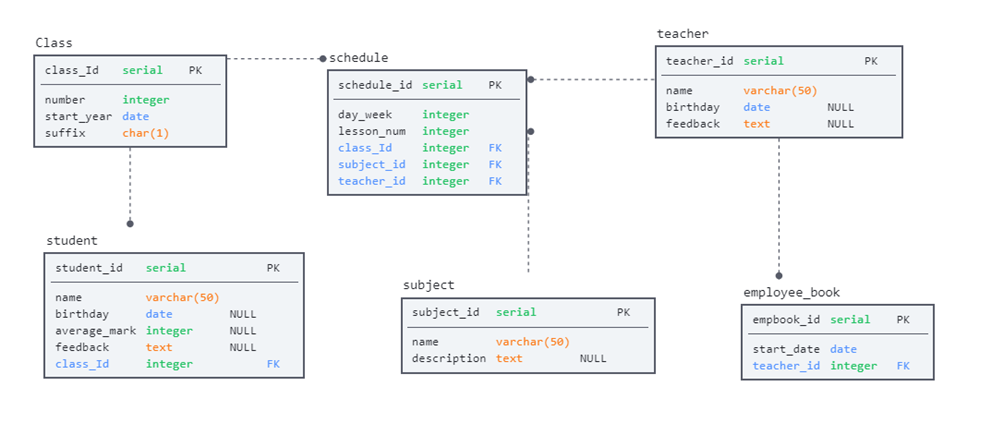
[Завдання 2. Пакетне генерування даних в таблиці Companies: 10](#_Toc57193394)

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

[Код програми 12](#_Toc57193396)

# Нормалізована модель даних БД «Школа»

На рисунку зображена нормалізована модель даних БД ‘Школа, розроблена на першій лабораторній роботі.

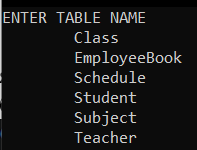


# Опис програми

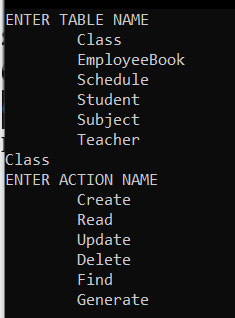
Програма створена для управління базою даних за допомогою базових операцій СУБД PostgreSQL і мови програмування **c#**, та реалізовує функціональні вимоги, що наведені у завданні. Додаток використовує шаблон проектування MVC: model – папка Model, View – папка view, Controller – папка Controller.

# Меню програми:

На зображенні показано головне меню програми з вибором таблиці, з якою хоче оперувати користувач. Потрібно ввести назву таблиці, з якою будуть проводитися маніпуляції.



Після вибору таблиці, відображається меню вибору операції.



Тепер можна писати назву операції, яку ми хочемо виконати з вибраною таблицею.

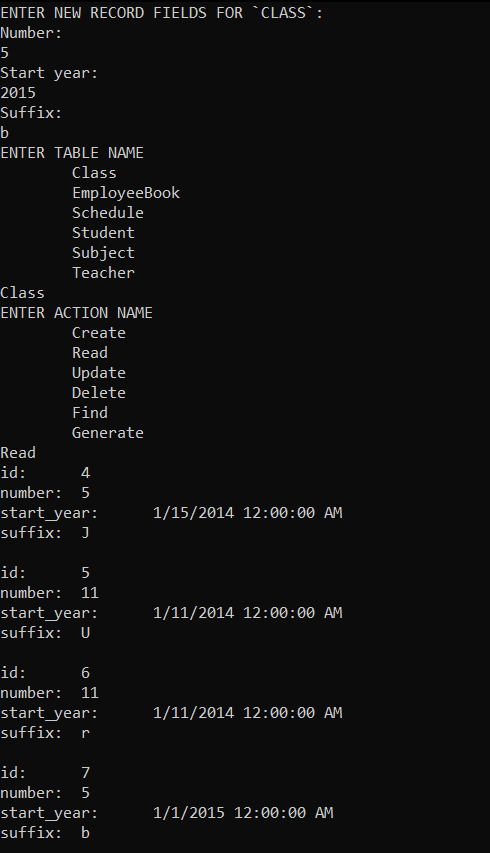
# Основні модулі програми

1. Program.cs – точка входу до програми, створює клас View і в ньому метод MainMenu;
2. View.cs – головне меню та виклик інших компонентів виду для кожної з таблиць;
3. IView.cs – інтерфейс, методи в якому дублюють назви операцій в субменю;
4. BaseView.cs – абстрактний клас, в якому є допоміжні методи, спільні для всіх видів;
5. ClassView.cs – реалізація IView, успадкована від BaseView для таблиці Class;
6. EmployeeBookView.cs – реалізація IView, успадкована від BaseView для таблиці EmployeeBook;
7. ScheduleView.cs – реалізація IView, успадкована від BaseView для таблиці Schedule;
8. StudentView.cs – реалізація IView, успадкована від BaseView для таблиці Student;
9. SubjectView.cs – реалізація IView, успадкована від BaseView для таблиці Subject;
10. TeacherView.cs – реалізація IView, успадкована від BaseView для таблиці Teacher;
11. IController.cs –інтерфейс, методи в якому дублюють назви операцій в субменю. Дані методи призначені для зв'язку рівня видів та моделі;
12. ClassController.cs – реалізація IController для таблиці Class;
13. EmployeeBookController.cs – реалізація IController для таблиці EmployeeBook;
14. ScheduleController.cs – реалізація IController для таблиці Schedule;
15. StudentController.cs – реалізація IController для таблиці Student;
16. SubjectController.cs – реалізація IController для таблиці Subject;
17. TeacherController.cs – реалізація IController для таблиці Teacher.
18. IModel.cs – інтерфейс, в якому описані методи, які реалізують CRUD: Create, Read, Update, Delete. За допомогою цих методів Контролер, в якому зберігається бізнес-логіка, реалізує інші методи, такі як Generate, Find.
19. BaseModel.cs – абстрактний клас, в якому є допоміжні методи, спільні для всіх моделей;
20. ClassModel.cs – реалізація IModel, успадкована від BaseModel для таблиці Class;
21. EmployeeBookModel.cs – реалізація IModel, успадкована від BaseModel для таблиці EmployeeBook;
22. ScheduleModel.cs – реалізація IModel, успадкована від BaseModel для таблиці Schedule;
23. StudentModel.cs – реалізація IModel, успадкована від BaseModel для таблиці Student;
24. SubjectModel.cs – реалізація IModel, успадкована від BaseModel для таблиці Subject;
25. TeacherView.cs – реалізація IView, успадкована від BaseView для таблиці Teacher;

# Завдання 1

## Додавання даних до БД:

Додали запис до таблиці Class і відразу подивилися результат. Новий запис автоматично отримав id = 7.

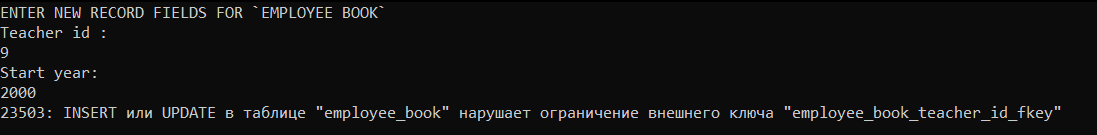


SQL-запит:

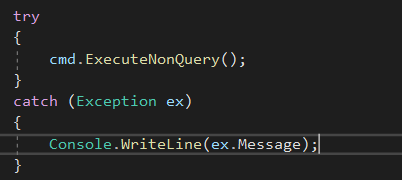
Insert into class(number, start\_year, suffix) VALUES(5, ‘2015-1-1’, ‘b’)

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

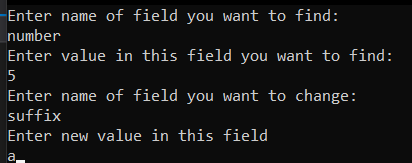
Розглянемо на прикладі EmployeeBook:



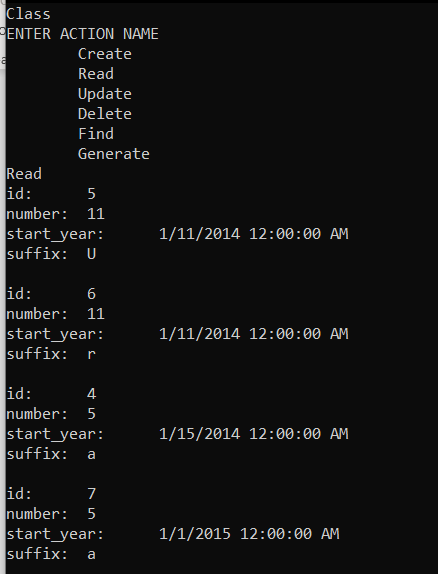
Код програми, який перехоплює помилку при створенні:



## Редагування даних:



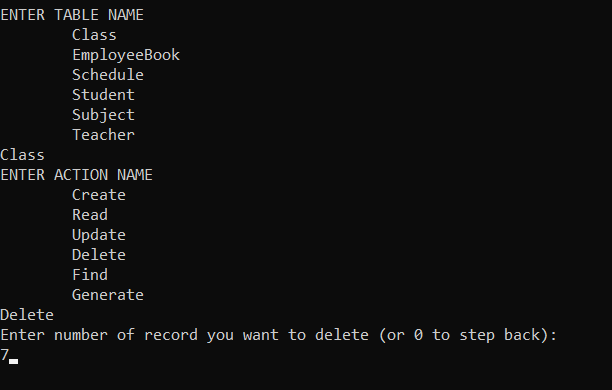
Результат:



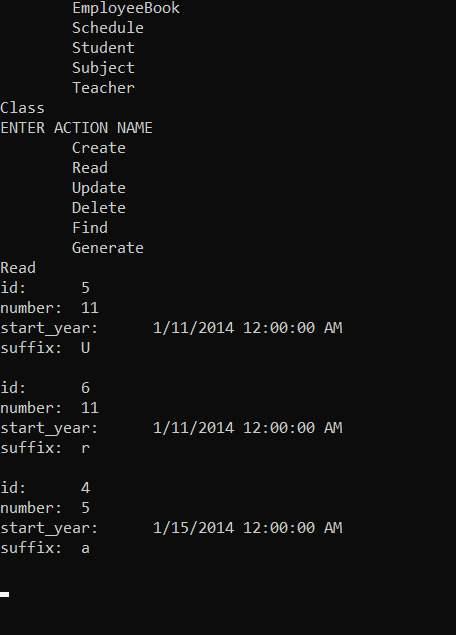
SQL-запит:

Update EmployeeBook set suffix = ‘b’ where number = 5

## Видалення даних:



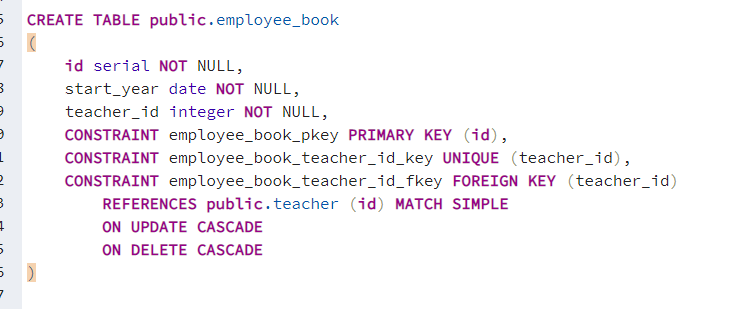
Результат:



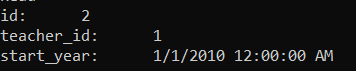
SQL-запит:

Delete from class where id = 7

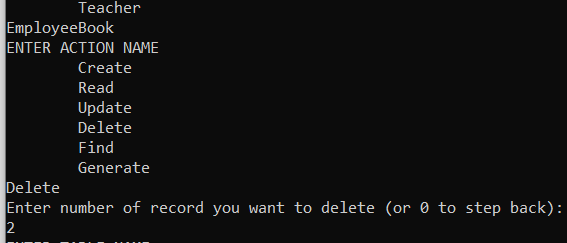
## Дослідження каскадного вилучення даних для таблиці EmployeeBook



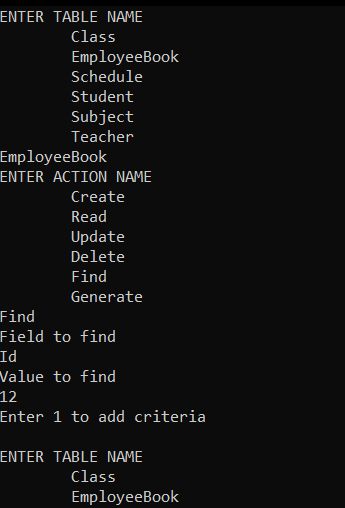
Візьмемо такий запис EmployeeBook:



Видалимо запис з EmployeeBook з Id = 2

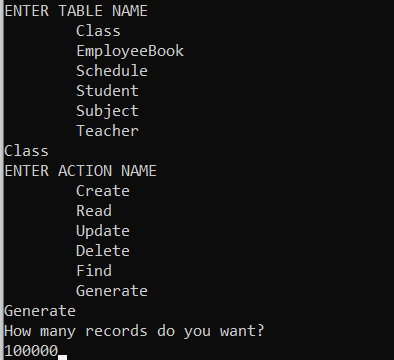


Відповідно запис, який ми взяли для прикладу з EmployeeBook (з Id = 2) повинен видалитися. Знайдемо такий запис:

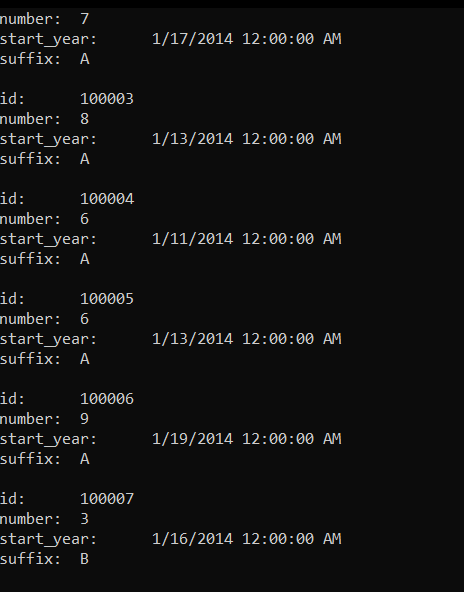
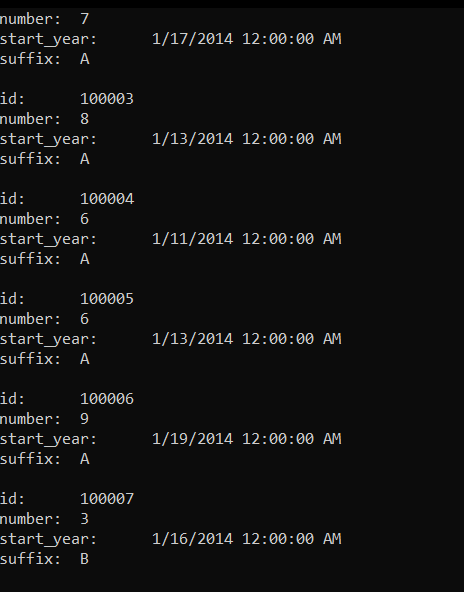


Це означає, що результатів не знайдено.

# Завдання 2. Пакетне генерування даних в таблиці Class:



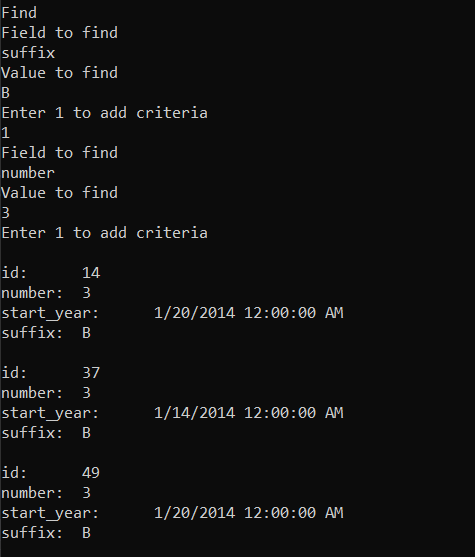
Результат (всі запити не можуть поміститися, тому лише останні дані):



SQL-запит:

"insert into class(number, start\_year, suffix) (select trunc(random()\*12)::int,timestamp '2014-01-10 20:00:00' + random() \* (timestamp '2014-01-20 20:00:00' - timestamp '2014-01-10 10:00:00'),chr(trunc(65 + random() \* 3)::int) from generate\_series(1, 100000) limit(100000))"

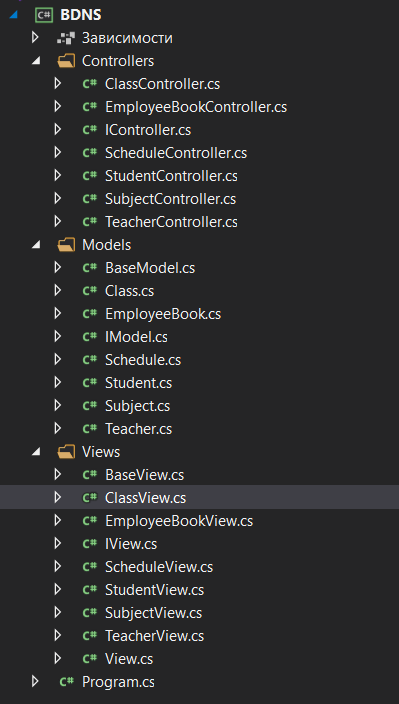
# Завдання 3. Пошук за двома-трьома атрибутами одночасно:



Відповідний sql-запит:

Select \* from class where suffix = ‘B’ and number = 3

# Код програми



## Program.cs

using BDNS.Views;

using System;

namespace BDNS

{

class Program

{

static void Main(string[] args)

{

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

var ConsoleView = new View(connectionString);

do

{

ConsoleView.MainMenu();

} while (true);

}

}

}

## View.cs

using System;

using System.Linq;

using System.Collections.Generic;

using System.Text;

namespace BDNS.Views

{

public class View

{

private string[] Tables = { "Class", "EmployeeBook", "Schedule", "Student", "Subject", "Teacher" };

private string[] Actions = { "Create", "Read", "Update", "Delete", "Find", "Generate" };

private string connectionString;

IView view = null;

public View(string connectionString)

{

this.connectionString = connectionString;

}

public void MainMenu()

{

string table = GetTableName();

switch (table)

{

case "Class":

view = new ClassView(connectionString);

break;

case "EmployeeBook":

view = new EmployeeBookView(connectionString);

break;

case "Schedule":

view = new ScheduleView(connectionString);

break;

case "Student":

view = new StudentView(connectionString);

break;

case "Subject":

view = new SubjectView(connectionString);

break;

case "Teacher":

view = new TeacherView(connectionString);

break;

}

string action = GetActionName();

switch (action)

{

case "Create":

view.Create();

break;

case "Read":

view.Read("");

break;

case "Update":

view.Update();

break;

case "Delete":

view.Delete();

break;

case "Find":

view.Find();

break;

case "Generate":

view.Generate();

break;

}

}

private string GetTableName()

{

string input = null;

do

{

Console.WriteLine("ENTER TABLE NAME");

foreach (var t in Tables)

{

Console.WriteLine("\t" + t);

}

input = Console.ReadLine();

if (!Tables.Contains(input))

{

Console.WriteLine("Input error: wrong table name.");

Console.ReadLine();

}

else

{

break;

}

} while (true);

return input;

}

private string GetActionName()

{

string input = null;

do

{

Console.WriteLine("ENTER ACTION NAME");

foreach (var t in Actions)

{

Console.WriteLine("\t"+ t);

}

input = Console.ReadLine();

if (!Actions.Contains(input))

{

Console.WriteLine("Input error: wrong action.");

Console.ReadLine();

}

else

{

break;

}

} while (true);

return input;

}

}

}

## IView

namespace BDNS.Views

{

public interface IView

{

public void Read(string whereCondition);

public void Create();

public void Delete()

{

}

public void Find();

public void Update();

public void Generate();

}

}

## BaseView

using System;

namespace BDNS.Views

{

public class BaseView

{

public int Delete()

{

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);

return id;

}

public string GetWhereCondition()

{

string[] fieldsToFind = new string[100];

string[] valuesToFind = new string[100];

int actualSize = 0;

for (int i = 0; i < 10; i++)

{

Console.WriteLine("Field to find");

fieldsToFind[i] = Console.ReadLine();

Console.WriteLine("Value to find");

valuesToFind[i] = Console.ReadLine();

Console.WriteLine("Enter 1 to add criteria");

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];

}

return whereCondition;

}

public string GetUpdateString()

{

string sqlUpdate, fieldToSet, valueToSet, fieldToFind, valueToFind;

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 + "'";

}

return "set " + fieldToSet + " = " + valueToSet + " where " + fieldToFind + " = " + valueToFind;

}

public int GetRecordsToGenerateAmount()

{

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

bool correct = false;

int recordsAmount;

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

return recordsAmount;

}

}

}

## ClassView

using BDNS.Controllers;

using System;

namespace BDNS.Views

{

public class ClassView : BaseView, IView

{

string connectionString;

ClassController controller;

public ClassView(string connectionString)

{

this.connectionString = connectionString;

this.controller = new ClassController(connectionString);

}

public ClassView()

{

}

public void Read(string whereCondition)

{

foreach (var g in controller.Read(whereCondition))

{

Console.WriteLine($"id:\t {g.id}");

Console.WriteLine($"number:\t {g.number}");

Console.WriteLine($"start\_year:\t {g.start\_year}");

Console.WriteLine($"suffix:\t {g.suffix}");

Console.WriteLine();

}

Console.ReadLine();

}

public void Create()

{

int id = 0;

int number = 0;

int start\_year = 0;

char suffix = 'f';

bool correct = false;

do

{

Console.Clear();

Console.WriteLine("ENTER NEW RECORD FIELDS FOR `CLASS`:");

Console.WriteLine("Number:");

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

if (correct == false)

{

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

Console.ReadLine();

continue;

}

Console.WriteLine("Start year:");

correct = Int32.TryParse(Console.ReadLine(), out start\_year);

if (correct == false)

{

Console.WriteLine("start year must be a number...");

Console.ReadLine();

continue;

}

Console.WriteLine("Suffix:");

string suffixString = Console.ReadLine();

suffix = suffixString[0];

} while (correct == false);

controller.Create(number, start\_year, suffix);

}

public void Delete()

{

int id = base.Delete();

controller.Delete(id);

}

public void Find()

{

this.Read(GetWhereCondition());

}

public void Update()

{

controller.Update(GetUpdateString());

}

public void Generate()

{

controller.Generate(GetRecordsToGenerateAmount());

}

}

}

## EmployeeBookView

using BDNS.Controllers;

using BDNS.Models;

using System;

using System.Collections.Generic;

using System.Text;

namespace BDNS.Views

{

public class EmployeeBookView : BaseView, IView

{

string connectionString;

EmployeeBookController controller;

public EmployeeBookView()

{

}

public EmployeeBookView(string connectionString)

{

this.connectionString = connectionString;

this.controller = new EmployeeBookController(connectionString);

}

public void Create()

{

string name;

int teacher\_id = 0;

int start\_year = 0;

bool correct = false;

do

{

Console.Clear();

Console.WriteLine("ENTER NEW RECORD FIELDS FOR `EMPLOYEE BOOK`");

Console.WriteLine("Teacher id :");

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

if (correct == false)

{

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

Console.ReadLine();

continue;

}

Console.WriteLine("Start year:");

correct = Int32.TryParse(Console.ReadLine(), out start\_year);

if (correct == false)

{

Console.WriteLine("start year must be a number...");

Console.ReadLine();

continue;

}

} while (correct == false);

controller.Create(teacher\_id, start\_year);

}

public void Delete()

{

int id = base.Delete();

controller.Delete(id);

}

public void Find()

{

controller.Find(GetWhereCondition());

}

public void Generate()

{

controller.Generate(GetRecordsToGenerateAmount());

}

public void Read(string whereCondition)

{

foreach(var g in controller.Read(whereCondition))

{

Console.WriteLine($"id:\t {g.id}");

Console.WriteLine($"teacher\_id:\t {g.teacher\_id}");

Console.WriteLine($"start\_year:\t {g.start\_year}");

Console.WriteLine();

}

Console.ReadLine();

}

public void Update()

{

controller.Update(GetUpdateString());

}

}

}

## ScheduleView

using BDNS.Controllers;

using BDNS.Models;

using System;

using System.Collections.Generic;

using System.Text;

namespace BDNS.Views

{

public class ScheduleView : BaseView, IView

{

string connectionString;

ScheduleController controller;

public ScheduleView(string connectionString)

{

this.connectionString = connectionString;

this.controller = new ScheduleController(connectionString);

}

public void Create()

{

int teacher\_id = 0;

int subject\_id = 0;

int lesson\_num = 0;

int day\_week = 0;

int class\_id = 0;

bool correct = false;

do

{

Console.Clear();

Console.WriteLine("ENTER NEW RECORD FIELDS FOR `SUBJECT`");

Console.WriteLine("Teacher id:");

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

if (correct == false)

{

Console.WriteLine("teacher-id must be a number...");

Console.ReadLine();

continue;

}

Console.WriteLine("Subject id:");

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

if (correct == false)

{

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

Console.ReadLine();

continue;

}

Console.WriteLine("class id:");

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

if (correct == false)

{

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

Console.ReadLine();

continue;

}

Console.WriteLine("day of week:");

correct = Int32.TryParse(Console.ReadLine(), out day\_week);

if (correct == false)

{

Console.WriteLine("day of week must be a number...");

Console.ReadLine();

continue;

}

Console.WriteLine("Number of lesson:");

correct = Int32.TryParse(Console.ReadLine(), out lesson\_num);

if (correct == false)

{

Console.WriteLine("lesson num must be a number...");

Console.ReadLine();

continue;

}

} while (correct == false);

controller.Create(teacher\_id, class\_id, subject\_id, day\_week, lesson\_num);

}

public void Delete()

{

int id = base.Delete();

controller.Delete(id);

}

public void Find()

{

controller.Find(GetWhereCondition());

}

public void Generate()

{

controller.Generate(GetRecordsToGenerateAmount());

}

public void Read(string whereCondition)

{

foreach (var g in controller.Read(whereCondition))

{

Console.WriteLine($"id:\t {g.id}");

Console.WriteLine($"teacher\_id:\t {g.teacher\_id}");

Console.WriteLine($"subject\_id:\t {g.subject\_id}");

Console.WriteLine($"lesson\_num:\t {g.lesson\_num}");

Console.WriteLine($"class\_id:\t {g.class\_id}");

Console.WriteLine($"day\_week:\t {g.day\_week}");

Console.WriteLine();

}

Console.ReadLine();

}

public void Update()

{

controller.Update(GetUpdateString());

}

}

}

## StudentView

using BDNS.Controllers;

using BDNS.Models;

using System;

using System.Collections.Generic;

using System.Text;

namespace BDNS.Views

{

public class StudentView : BaseView, IView

{

string connectionString;

StudentController controller;

public StudentView(string connectionString)

{

this.connectionString = connectionString;

this.controller = new StudentController(connectionString);

}

public void Read(string whereCondition)

{

foreach (var g in controller.Read(whereCondition))

{

Console.WriteLine($"id:\t {g.id}");

Console.WriteLine($"class\_id:\t {g.class\_id}");

Console.WriteLine($"birthday:\t {g.birthday}");

Console.WriteLine($"average\_mark:\t {g.average\_mark}");

Console.WriteLine($"feedback:\t {g.feedback}");

Console.WriteLine($"name:\t {g.name}");

Console.WriteLine();

}

Console.ReadLine();

}

public void Create()

{

string name;

int class\_id = 0;

DateTime birthday = new DateTime();

decimal average\_mark = 0;

string feedback = null;

bool correct = false;

do

{

Console.Clear();

Console.WriteLine("ENTER NEW RECORD FIELDS FOR `STUDENT`");

Console.WriteLine("Name:");

name = Console.ReadLine();

if (name.Length > 50)

{

correct = false;

Console.WriteLine("Length of name > 50. It is wrong.");

Console.ReadLine();

continue;

}

Console.WriteLine("Feedback:");

feedback = Console.ReadLine();

Console.WriteLine("Birthday:");

int year, month, day;

Console.WriteLine("Year:");

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

if (correct == false)

{

Console.WriteLine("year must be a number...");

Console.ReadLine();

continue;

}

Console.WriteLine("Month:");

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

if (correct == false)

{

Console.WriteLine("month must be a number...");

Console.ReadLine();

continue;

}

Console.WriteLine("Day:");

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

if (correct == false)

{

Console.WriteLine("day must be a number...");

Console.ReadLine();

continue;

}

try

{

birthday = new DateTime(year, month, day);

}

catch(Exception ex)

{

Console.WriteLine(ex.Message);

}

Console.WriteLine("Average Mark:");

correct = Decimal.TryParse(Console.ReadLine(), out average\_mark);

if (correct == false)

{

Console.WriteLine("average mark must be a number...");

Console.ReadLine();

continue;

}

Console.WriteLine("Class id:");

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

if (correct == false)

{

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

Console.ReadLine();

continue;

}

} while (correct == false);

controller.Create(name, average\_mark, birthday, feedback, class\_id);

}

public void Delete()

{

int id = base.Delete();

controller.Delete(id);

}

public void Find()

{

controller.Find(GetWhereCondition());

}

public void Update()

{

controller.Update(GetUpdateString());

}

public void Generate()

{

controller.Generate(GetRecordsToGenerateAmount());

}

}

}

## SubjectView

using BDNS.Controllers;

using BDNS.Models;

using System;

using System.Collections.Generic;

using System.Text;

namespace BDNS.Views

{

public class SubjectView : BaseView, IView

{

string connectionString;

SubjectController controller;

public SubjectView(string connectionString)

{

this.connectionString = connectionString;

this.controller = new SubjectController(connectionString);

}

public void Read(string whereCondition)

{

foreach (var g in controller.Read(whereCondition))

{

Console.WriteLine($"id:\t {g.id}");

Console.WriteLine($"name:\t {g.name}");

Console.WriteLine($"description:\t {g.description}");

Console.WriteLine();

}

Console.ReadLine();

}

public void Create() {

string name;

string description = null;

bool correct = false;

do

{

Console.Clear();

Console.WriteLine("ENTER NEW RECORD FIELDS FOR `SUBJECT`");

Console.WriteLine("Name:");

name = Console.ReadLine();

if (name.Length > 50)

{

correct = false;

Console.WriteLine("Length of name > 50. It is wrong.");

Console.ReadLine();

continue;

}

Console.WriteLine("Description:");

description = Console.ReadLine();

} while (correct == false);

controller.Create(name, description);

}

public void Delete()

{

int id = base.Delete();

controller.Delete(id);

}

public void Find()

{

controller.Find(GetWhereCondition());

}

public void Update()

{

controller.Update(GetUpdateString());

}

public void Generate()

{

controller.Generate(GetRecordsToGenerateAmount());

}

}

}

## TeacherView

using BDNS.Controllers;

using BDNS.Models;

using System;

using System.Collections.Generic;

using System.Text;

namespace BDNS.Views

{

public class SubjectView : BaseView, IView

{

string connectionString;

SubjectController controller;

public SubjectView(string connectionString)

{

this.connectionString = connectionString;

this.controller = new SubjectController(connectionString);

}

public void Read(string whereCondition)

{

foreach (var g in controller.Read(whereCondition))

{

Console.WriteLine($"id:\t {g.id}");

Console.WriteLine($"name:\t {g.name}");

Console.WriteLine($"description:\t {g.description}");

Console.WriteLine();

}

Console.ReadLine();

}

public void Create() {

string name;

string description = null;

bool correct = false;

do

{

Console.Clear();

Console.WriteLine("ENTER NEW RECORD FIELDS FOR `SUBJECT`");

Console.WriteLine("Name:");

name = Console.ReadLine();

if (name.Length > 50)

{

correct = false;

Console.WriteLine("Length of name > 50. It is wrong.");

Console.ReadLine();

continue;

}

Console.WriteLine("Description:");

description = Console.ReadLine();

} while (correct == false);

controller.Create(name, description);

}

public void Delete()

{

int id = base.Delete();

controller.Delete(id);

}

public void Find()

{

controller.Find(GetWhereCondition());

}

public void Update()

{

controller.Update(GetUpdateString());

}

public void Generate()

{

controller.Generate(GetRecordsToGenerateAmount());

}

}

}

## IController

using BDNS.Models;

using System.Collections.Generic;

namespace BDNS.Controllers

{

public interface IController<T> where T: BaseModel

{

public List<T> Read(string whereCondition);

public void Delete(int id);

public void Update(string updateString);

public void Find(string whereCondition);

public void Generate(int recordsAmount);

}

}

## ClassController

using BDNS.Models;

using System.Collections.Generic;

namespace BDNS.Controllers

{

public class ClassController : IController<Class>

{

string connectionString;

Class model;

public ClassController(string connectionString)

{

this.connectionString = connectionString;

model = new Class(connectionString);

}

public void Create(int number, int start\_year, char suffix)

{

model.Create(number, start\_year, suffix);

}

public List<Class> Read(string whereCondition)

{

return model.Read(whereCondition);

}

public void Delete(int id)

{

model.Delete(id);

}

public void Update(string updateString)

{

model.Update(updateString);

}

public void Find(string whereCondition)

{

model.Read(whereCondition);

}

public void Generate(int recordsAmount)

{

model.Create(recordsAmount);

}

}

}

## EmployeeBookController

using BDNS.Models;

using System;

using System.Collections.Generic;

using System.Text;

namespace BDNS.Controllers

{

public class EmployeeBookController : IController<EmployeeBook>

{

string connectionString;

EmployeeBook model;

public EmployeeBookController(string connectionString)

{

this.connectionString = connectionString;

model = new EmployeeBook(connectionString);

}

public void Create(int start\_year, int teacher\_id)

{

model.Create(start\_year, teacher\_id);

}

public void Delete(int id)

{

model.Delete(id);

}

public void Update(string updateString)

{

model.Update(updateString);

}

public void Find(string whereCondition)

{

model.Read(whereCondition);

}

public void Generate(int recordsAmount)

{

model.Create(recordsAmount);

}

public List<EmployeeBook> Read(string whereCondition)

{

return model.Read(whereCondition);

}

}

}

## ScheduleController

using BDNS.Models;

using System;

using System.Collections.Generic;

using System.Text;

namespace BDNS.Controllers

{

public class EmployeeBookController : IController<EmployeeBook>

{

string connectionString;

EmployeeBook model;

public EmployeeBookController(string connectionString)

{

this.connectionString = connectionString;

model = new EmployeeBook(connectionString);

}

public void Create(int start\_year, int teacher\_id)

{

model.Create(start\_year, teacher\_id);

}

public void Delete(int id)

{

model.Delete(id);

}

public void Update(string updateString)

{

model.Update(updateString);

}

public void Find(string whereCondition)

{

model.Read(whereCondition);

}

public void Generate(int recordsAmount)

{

model.Create(recordsAmount);

}

public List<EmployeeBook> Read(string whereCondition)

{

return model.Read(whereCondition);

}

}

}

## StudentController

using BDNS.Models;

using System;

using System.Collections.Generic;

using System.Text;

namespace BDNS.Controllers

{

public class StudentController : IController<Student>

{

string connectionString;

Student model;

public StudentController(string connectionString)

{

this.connectionString = connectionString;

model = new Student(connectionString);

}

public void Create(string name, decimal average\_mark, DateTime birthday, string feedback, int class\_id)

{

model.Create(name, average\_mark, birthday, feedback, class\_id);

}

public void Delete(int id)

{

model.Delete(id);

}

public void Update(string updateString)

{

model.Update(updateString);

}

public void Find(string whereCondition)

{

model.Read(whereCondition);

}

public void Generate(int recordsAmount)

{

model.Create(recordsAmount);

}

public List<Student> Read(string whereCondition)

{

throw new NotImplementedException();

}

}

}

## SubjectController

using BDNS.Models;

using System;

using System.Collections.Generic;

using System.Text;

namespace BDNS.Controllers

{

public class SubjectController : IController<Subject>

{

string connectionString;

Subject model;

public SubjectController(string connectionString)

{

this.connectionString = connectionString;

model = new Subject(connectionString);

}

public void Create(string name, string description)

{

model.Create(name, description);

}

public List<Subject> Read(string whereCondition)

{

return model.Read(whereCondition);

}

public void Delete(int id)

{

model.Delete(id);

}

public void Update(string updateString)

{

model.Update(updateString);

}

public void Find(string whereCondition)

{

model.Read(whereCondition);

}

public void Generate(int recordsAmount)

{

model.Create(recordsAmount);

}

}

}

## TeacherController

using BDNS.Models;

using System;

using System.Collections.Generic;

using System.Text;

namespace BDNS.Controllers

{

public class TeacherController : IController<Teacher>

{

string connectionString;

Teacher model;

public TeacherController(string connectionString)

{

this.connectionString = connectionString;

model = new Teacher(connectionString);

}

public void Create(string name, DateTime birthday, string feedback)

{

model.Create(name, birthday, feedback);

}

public List<Teacher> Read(string whereCondition)

{

return model.Read(whereCondition);

}

public void Delete(int id)

{

model.Delete(id);

}

public void Update(string updateString)

{

model.Update(updateString);

}

public void Find(string whereCondition)

{

model.Read(whereCondition);

}

public void Generate(int recordsAmount)

{

model.Create(recordsAmount);

}

}

}

## BaseModel

using Npgsql;

using System;

namespace BDNS.Models

{

public abstract class BaseModel

{

public string connectionString;

protected NpgsqlConnection sqlConnection;

public BaseModel(string connectionString)

{

this.connectionString = connectionString;

this.sqlConnection = new NpgsqlConnection(connectionString);

}

protected void Delete(string sqlDelete, int id)

{

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();

}

}

public void CreateRandom(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();

}

}

public void Update(string sqlUpdate)

{

sqlConnection.Open();

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

try

{

cmd.Prepare();

cmd.ExecuteNonQuery();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.ReadLine();

}

finally

{

sqlConnection.Close();

}

}

}

}

## IModel

using System;

using System.Collections.Generic;

using System.Text;

namespace BDNS.Models

{

public interface IModel<T> where T: BaseModel

{

public List<T> Read(string whereCondition);

public void Delete(int id);

public void Update(string updateString);

public void Create(int recordsAmount);

}

}

## EmployeeBook

using Npgsql;

using System;

using System.Collections.Generic;

using System.Text;

namespace BDNS.Models

{

public class EmployeeBook : BaseModel, IModel<EmployeeBook>

{

public object id { get; set; }

public object start\_year { get; set; }

public object teacher\_id { get; set; }

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

public void Create(int start\_year, int teacher\_id)

{

string sqlInsert = "Insert into employee\_book(start\_year, teacher\_id) VALUES(@start\_year, @teacher\_id)";

sqlConnection.Open();

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

cmd.Parameters.AddWithValue("start\_year", new DateTime(start\_year, 1, 1));

cmd.Parameters.AddWithValue("teacher\_id", teacher\_id);

cmd.Prepare();

try

{

cmd.ExecuteNonQuery();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.ReadLine();

}

finally

{

sqlConnection.Close();

}

}

public List<EmployeeBook> Read(string whereCondition)

{

sqlConnection.Open();

string sqlSelect = "select id, start\_year, teacher\_id from employee\_book" + whereCondition;

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

List<EmployeeBook> list = new List<EmployeeBook>();

try

{

using NpgsqlDataReader rdr = cmd.ExecuteReader();

while (rdr.Read())

{

var e = new EmployeeBook(connectionString);

e.id = rdr.GetValue(0);

e.start\_year = rdr.GetValue(1);

e.teacher\_id = rdr.GetValue(2);

list.Add(e);

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.ReadLine();

}

finally

{

sqlConnection.Close();

}

return list;

}

public void Delete(int id)

{

base.Delete("delete from employee\_book where id = ", id);

}

public void Update(string updateString)

{

base.Update("Update employee\_book " + updateString);

}

public void Create(int recordsAmount)

{

string sqlGenerate = "insert into employee\_book(start\_year, teacher\_id) (select "

+ "timestamp '2014-01-10 20:00:00' + random() \* (timestamp '2014-01-20 20:00:00' - timestamp '2014-01-10 10:00:00')," // start\_year

+ $" teacher.id from generate\_series(1, {recordsAmount}), teacher limit({recordsAmount}))";

base.CreateRandom(sqlGenerate);

}

}

}

## Class

using Npgsql;

using System;

using System.Collections.Generic;

using System.Text;

namespace BDNS.Models

{

public class Class : BaseModel, IModel<Class>

{

public object id { get; set; }

public object number { get; set; }

public object start\_year { get; set; }

public object suffix { get; set; }

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

public void Create(int number, int start\_year, char suffix)

{

string sqlInsert = "Insert into class(number, start\_year, suffix) VALUES(@number, @start\_year, @suffix)";

sqlConnection.Open();

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

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

cmd.Parameters.AddWithValue("start\_year", new DateTime(start\_year,1,1));

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

cmd.Prepare();

try

{

cmd.ExecuteNonQuery();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.ReadLine();

}

finally

{

sqlConnection.Close();

}

}

public List<Class> Read(string whereCondition)

{

sqlConnection.Open();

string sqlSelect = "select id, number, start\_year, suffix from class" + whereCondition;

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

List<Class> list= new List<Class>();

try

{

using NpgsqlDataReader rdr = cmd.ExecuteReader();

while (rdr.Read())

{

var e = new Class(connectionString);

e.id= rdr.GetValue(0);

e.number = rdr.GetValue(1);

e.start\_year = rdr.GetValue(2);

e.suffix = rdr.GetValue(3);

list.Add(e);

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.ReadLine();

}

finally

{

sqlConnection.Close();

}

return list;

}

public void Delete(int id)

{

base.Delete("delete from class where id = ", id);

}

public void Update(string updateString)

{

base.Update("Update class " + updateString);

}

public void Create(int recordsAmount)

{

string sqlGenerate = "insert into class(number, start\_year, suffix) (select "

+ "trunc(random()\*12)::int," // number

+ "timestamp '2014-01-10 20:00:00' + random() \* (timestamp '2014-01-20 20:00:00' - timestamp '2014-01-10 10:00:00')," // start\_year

+ "chr(trunc(65 + random() \* 3)::int) " // suffix

+ $" from generate\_series(1, {recordsAmount}) limit({recordsAmount}))";

base.CreateRandom(sqlGenerate);

}

}

}

## Schedule

using Npgsql;

using System;

using System.Collections.Generic;

using System.Text;

namespace BDNS.Models

{

public class Schedule : BaseModel, IModel<Schedule>

{

public object id { get; set; }

public object teacher\_id { get; set; }

public object class\_id { get; set; }

public object subject\_id { get; set; }

public object day\_week { get; set; }

public object lesson\_num { get; set; }

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

public void Create(int teacher\_id, int class\_id, int subject\_id, int day\_week, int lesson\_num)

{

string sqlInsert = "Insert into schedule(teacher\_id, class\_id, subject\_id, day\_week, lesson\_num) VALUES(@teacher\_id, @class\_id, @subject\_id, @day\_week, @lesson\_num)";

sqlConnection.Open();

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

cmd.Parameters.AddWithValue("teacher\_id", teacher\_id);

cmd.Parameters.AddWithValue("class\_id", class\_id);

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

cmd.Parameters.AddWithValue("day\_week", day\_week);

cmd.Parameters.AddWithValue("lesson\_num", lesson\_num);

cmd.Prepare();

try

{

cmd.ExecuteNonQuery();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.ReadLine();

}

finally

{

sqlConnection.Close();

}

}

public List<Schedule> Read(string whereCondition)

{

sqlConnection.Open();

string sqlSelect = "select id, teacher\_id, class\_id, subject\_id, day\_week, lesson\_num from schedule" + whereCondition;

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

List<Schedule> list = new List<Schedule>();

try

{

using NpgsqlDataReader rdr = cmd.ExecuteReader();

while (rdr.Read())

{

var e = new Schedule(connectionString);

e.id = rdr.GetValue(0);

e.teacher\_id = rdr.GetValue(1);

e.class\_id = rdr.GetValue(2);

e.day\_week = rdr.GetValue(3);

e.lesson\_num = rdr.GetValue(1);

e.class\_id = rdr.GetValue(2);

list.Add(e);

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.ReadLine();

}

finally

{

sqlConnection.Close();

}

return list;

}

public void Delete(int id)

{

base.Delete("delete from schedule where id = ", id);

}

public void Update(string updateString)

{

base.Update("Update class " + updateString);

}

public void Create(int recordsAmount)

{

string sqlGenerate = "insert into schedule(teacher\_id, class\_id, subject\_id, day\_week, lesson\_num) (select "

+ "teacher.id, class.id, " // teacher\_id, class\_id

+ "trunc(random()\*7)::int," // day\_week

+ "trunc(random()\*8)::int," // lesson\_num

+ $" from teacher, class, generate\_series(1, {recordsAmount}) limit({recordsAmount}))";

base.CreateRandom(sqlGenerate);

}

}

}

## Student

using Npgsql;

using System;

using System.Collections.Generic;

using System.Text;

namespace BDNS.Models

{

public class Student : BaseModel, IModel<Student>

{

public object id { get; set; }

public object name { get; set; }

public object average\_mark { get; set; }

public object birthday { get; set; }

public object feedback { get; set; }

public object class\_id { get; set; }

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

public void Create(string name, decimal average\_mark, DateTime birthday, string feedback, int class\_id)

{

string sqlInsert = "Insert into student(name, average\_mark, birthday, feedback, class\_id) VALUES(@name, @average\_mark, @birthday, @feedback, @class\_id)";

sqlConnection.Open();

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

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

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

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

cmd.Parameters.AddWithValue("average\_mark", average\_mark);

cmd.Parameters.AddWithValue("class\_id", class\_id);

cmd.Prepare();

try

{

cmd.ExecuteNonQuery();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.ReadLine();

}

finally

{

sqlConnection.Close();

}

}

public List<Student> Read(string whereCondition)

{

sqlConnection.Open();

string sqlSelect = "select id, name, average\_mark, birthday, feedback, class\_id from student" + whereCondition;

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

List<Student> list = new List<Student>();

try

{

using NpgsqlDataReader rdr = cmd.ExecuteReader();

while (rdr.Read())

{

var e = new Student(connectionString);

e.id = rdr.GetValue(0);

e.name = rdr.GetValue(1);

e.average\_mark = rdr.GetValue(2);

e.birthday = rdr.GetValue(3);

e.feedback = rdr.GetValue(4);

e.class\_id = rdr.GetValue(5);

list.Add(e);

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.ReadLine();

}

finally

{

sqlConnection.Close();

}

return list;

}

public void Delete(int id)

{

base.Delete("delete from student where id = ", id);

}

public void Update(string updateString)

{

base.Update("Update class " + updateString);

}

public void Create(int recordsAmount)

{

string sqlGenerate = "insert into student(name, average\_mark, birthday, feedback, class\_id) (select "

+ "chr(trunc(65 + random() \* 50)::int) || chr(trunc(65 + random() \* 25)::int)," // name

+ "timestamp '2014-01-10 20:00:00' + random() \* (timestamp '2014-01-20 20:00:00' - timestamp '2014-01-10 10:00:00')," // birthday

+ "trunc(random()\*1000)::int," // average\_mark

+ "chr(trunc(65 + random() \* 50)::int) || chr(trunc(65 + random() \* 25)::int)," // feedback

+ "class.id" // class\_id

+ $" from generate\_series(1, {recordsAmount}), class limit({recordsAmount}))";

base.CreateRandom(sqlGenerate);

}

}

}

## Subject

using Npgsql;

using System;

using System.Collections.Generic;

using System.Text;

namespace BDNS.Models

{

public class Subject : BaseModel, IModel<Subject>

{

public object id { get; set; }

public object name { get; set; }

public object description { get; set; }

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

public void Create( string name, string description)

{

string sqlInsert = "Insert into subject(name, description) VALUES(@name, @description)";

sqlConnection.Open();

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

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

cmd.Parameters.AddWithValue("birthday", description);

cmd.Prepare();

try

{

cmd.ExecuteNonQuery();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

finally

{

sqlConnection.Close();

}

}

public List<Subject> Read(string whereCondition)

{

sqlConnection.Open();

string sqlSelect = "select id, name, description from subject" + whereCondition;

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

List<Subject> list = new List<Subject>();

try

{

using NpgsqlDataReader rdr = cmd.ExecuteReader();

while (rdr.Read())

{

var e = new Subject(connectionString);

e.id = rdr.GetValue(0);

e.name = rdr.GetValue(1);

e.description = rdr.GetValue(2);

list.Add(e);

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.ReadLine();

}

finally

{

sqlConnection.Close();

}

return list;

}

public void Delete(int id)

{

base.Delete("delete from subject where id = ", id);

}

public void Update(string updateString)

{

base.Update("Update class " + updateString);

}

public void Create(int recordsAmount)

{

string sqlGenerate = "insert into subject(name, description) (select "

+ "chr(trunc(65 + random() \* 50)::int) || chr(trunc(65 + random() \* 25)::int), " // name

+ "chr(trunc(65 + random() \* 50)::int) || chr(trunc(65 + random() \* 25)::int) " // description

+ $" from generate\_series(1, {recordsAmount}) limit({recordsAmount}))";

base.CreateRandom(sqlGenerate);

}

}

}

## Teacher

using Npgsql;

using System;

using System.Collections.Generic;

namespace BDNS.Models

{

public class Teacher : BaseModel, IModel<Teacher>

{

public object id { get; set; }

public object name { get; set; }

public object birthday { get; set; }

public object feedback { get; set; }

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

public void Create(string name, DateTime birthday, string feedback)

{

string sqlInsert = "Insert into teacher(name, birthday, feedback) VALUES(@name, @birthday, @feedback)";

sqlConnection.Open();

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

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

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

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

cmd.Prepare();

try

{

cmd.ExecuteNonQuery();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.ReadLine();

}

finally

{

sqlConnection.Close();

}

}

public List<Teacher> Read(string whereCondition)

{

sqlConnection.Open();

string sqlSelect = "select id, name, birthday, feedback from teacher" + whereCondition;

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

List<Teacher> list = new List<Teacher>();

try

{

using NpgsqlDataReader rdr = cmd.ExecuteReader();

while (rdr.Read())

{

var e = new Teacher(connectionString);

e.id = rdr.GetValue(0);

e.name = rdr.GetValue(1);

e.birthday = rdr.GetValue(2);

e.feedback = rdr.GetValue(3);

list.Add(e);

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.ReadLine();

}

finally

{

sqlConnection.Close();

}

return list;

}

public void Delete(int id)

{

base.Delete("delete from teacher where id = ", id);

}

public void Update(string updateString)

{

base.Update("Update class " + updateString);

}

public void Create(int recordsAmount)

{

string sqlGenerate = "insert into teacher(name, birthday, feedback) (select "

+ "chr(trunc(65 + random() \* 50)::int) || chr(trunc(65 + random() \* 25)::int)," // name

+ "timestamp '2014-01-10 20:00:00' + random() \* (timestamp '2014-01-20 20:00:00' - timestamp '2014-01-10 10:00:00')," // birthday

+ "chr(trunc(65 + random() \* 50)::int) || chr(trunc(65 + random() \* 25)::int) " // feedback

+ $" from generate\_series(1, {recordsAmount}) limit({recordsAmount}))";

base.CreateRandom(sqlGenerate);

}

}

}