



НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування та спеціалізованих комп'ютерних  
систем**

**Розрахунково-графічна робота**

**з дисципліни Баз даних і засоби управління**

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

Виконав:

Студент групи КВ-33

Ткаченко В.В.

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

### **Виконання роботи**

Нижче наведені сутності предметної області «Система обліку відвідуваності на заняттях університету» та зв'язки між ними.

#### **Сутності предметної області**

Для побудови бази даних обраної області, були виділені такі сутності:

1. **Група студентів (Student\_group)** – представляє академічні групи:
  - Атрибути: id групи, назва групи, факультет.
2. **Студент (Student)** – представляє окремого студента:
  - Атрибути: id студента, ім'я, прізвище, електронна пошта, id групи.
3. **Викладач (Teacher)** – представляє викладачів університету:
  - Атрибути: id викладача, ім'я, прізвище.
4. **Предмет (Subject)** – представляє навчальні дисципліни:
  - Атрибути: id предмета, назва предмета.
5. **Призначення викладача (Teacher\_Subject)** – відображає, які викладачі ведуть які предмети:
  - Атрибути: id викладача, id предмета.
6. **Заняття (Class)** – представляє окреме проведене заняття:
  - Атрибути: id заняття, id предмета, id викладача, дата та час заняття, тип заняття (лекція, практика тощо).
7. **Відвідуваність (Attendance)** – представляє факт відвідування студентом конкретного заняття:
  - Атрибути: id запису, id заняття, id студента, статус (присутній, відсутній, запізнився).

#### **Зв'язки між сутностями предметної області**

Зв'язок «Група студентів» – «Студент»:

- Тип зв'язку: 1 до N (одна група може містити багато студентів; один студент належить лише одній групі).

Зв'язок «Студент» - «Заняття»:

- Тип зв'язку: M до N (один студент може відвідувати багато занять; одне заняття може відвідати багато студентів).

Зв'язок «Предмет» - «Заняття»:

– Тип зв'язку: 1 до N (один предмет може мати багато занять; одне заняття належить лише одному предмету).

Зв'язок «Викладач» - «Заняття»:

– Тип зв'язку: 1 до N (один викладач може проводити багато занять; одне заняття веде лише один викладач).

Зв'язок «Викладач» - «Предмет»:

– Тип зв'язку: М до N (один викладач може викладати багато предметів, і один предмет може викладатися різними викладачами).

Графічне подання концептуальної моделі «Сутність-зв'язок» зображено на рисунку 1

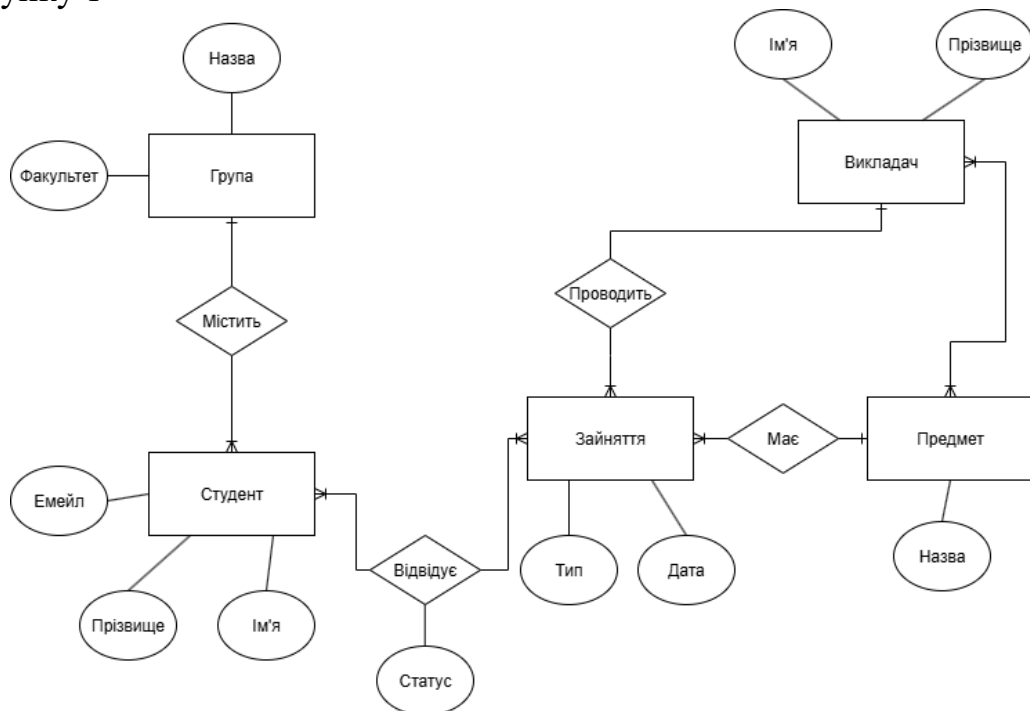


Рисунок 1 – ER-діаграма, побудована за нотацією "Пташина лапка"

Графічне подання логічної моделі «Сутність-зв'язок» зображено на рисунку 2

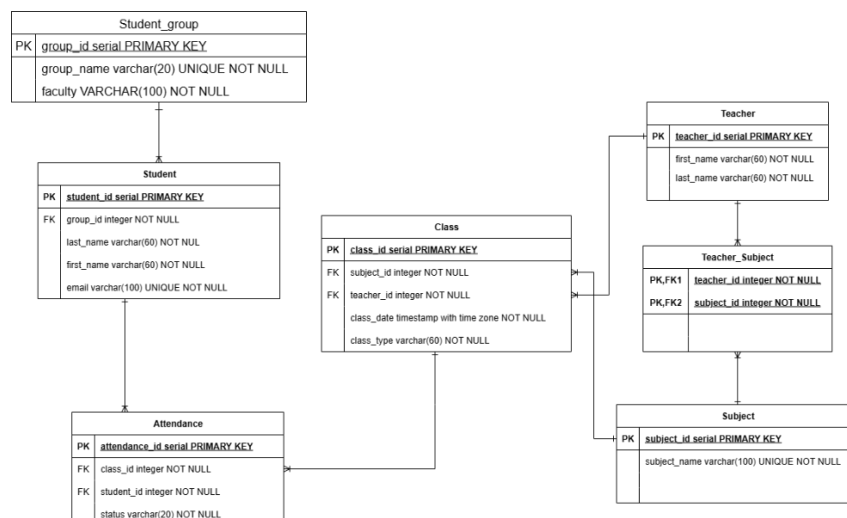


Рисунок 2 – Схема бази даних

## Середовище та компоненти розробки

У процесі розробки була використана мова програмування C++, середовище розробки Visual Studio Code, а також була використані бібліотеки pqxx, SFML та ImGui. Pqxx надає API для взаємодії з базою даних PostgreSQL, а SFML та ImGui надають прості засоби створення вікна та інтерфейсу.

## Шаблон проектування

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

1. Модель (Model) – це клас, що відображає логіку роботи з даними, яку не бачить користувач. Він обробляє всі операції з даними, такі як додавання, оновлення і тд.
2. Представлення (View) – це клас, через який користувач взаємодіє з програмою. Відповідає за графічне виведення даних, але не керує фізичними даними.
3. Контролер (Controller) – це клас, який відповідає за зв'язок між користувачем і системою. Він приймає введені користувачем дані та обробляє їх. В залежності від результатів, викликає відповідні дії з Model або View.

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

## Структура програми та її опис

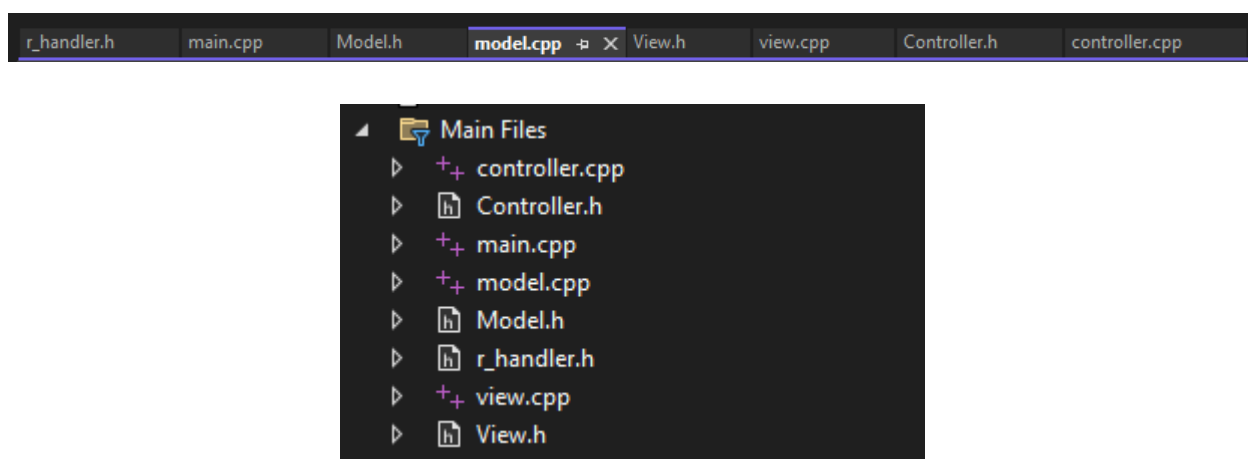


Рисунок 3 – Структура програми

У файлі main.cpp відбувається створення об'єкту класу контроллер, який автоматично бере управління на себе.

У модулі Model (Model.h та model.cpp) описаний клас моделі, який відповідає за управління підключенням до бази даних і виконанням низькорівневих безпечних запитів до неї.

У модулі Controller реалізовані команди взаємодії з користувачем,

включаючи обробку запитів користувача, виконання пошуку, а також інші дії, необхідні для взаємодії з моделлю та представленням.

У модулі View описаний клас, який відображає результати виконання різних дій користувача у головному вікні. Цей компонент відповідає за представлення даних користувачу в зручному для сприйняття вигляді.

У додатковому файлі `r_handler.cpp` описаний простий клас, для передачі повідомлень помилок.

Отже, структура програми відповідає патерну MVC.

## Структура меню програми

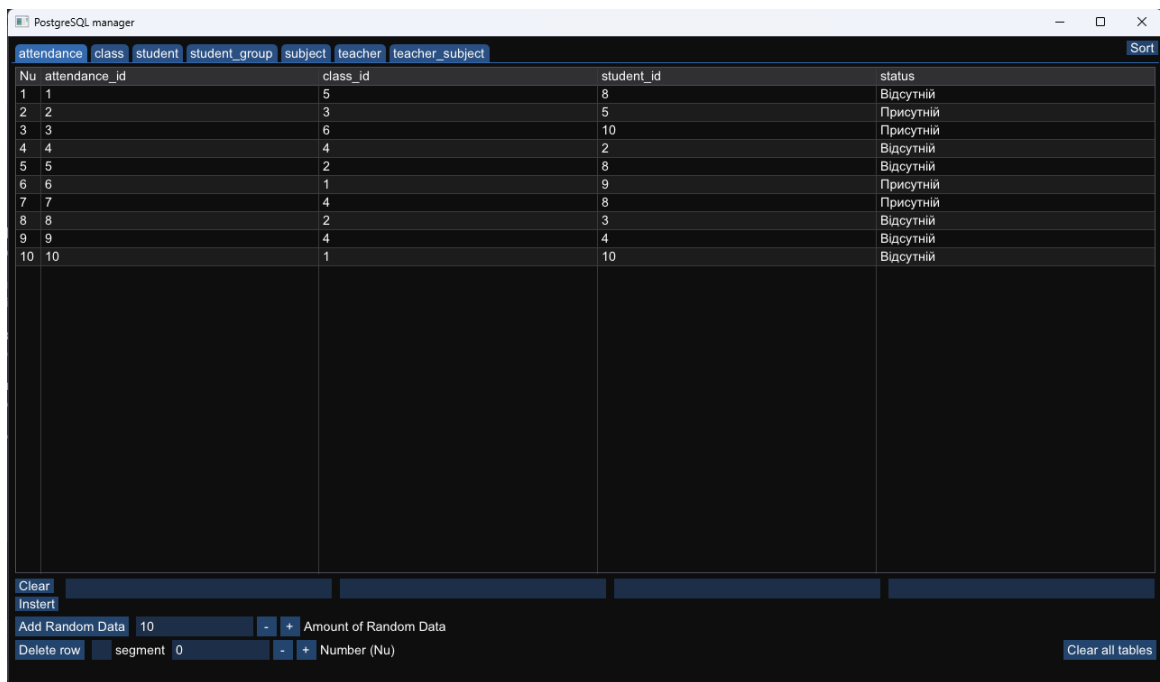


Рисунок 4 – Структура меню користувача

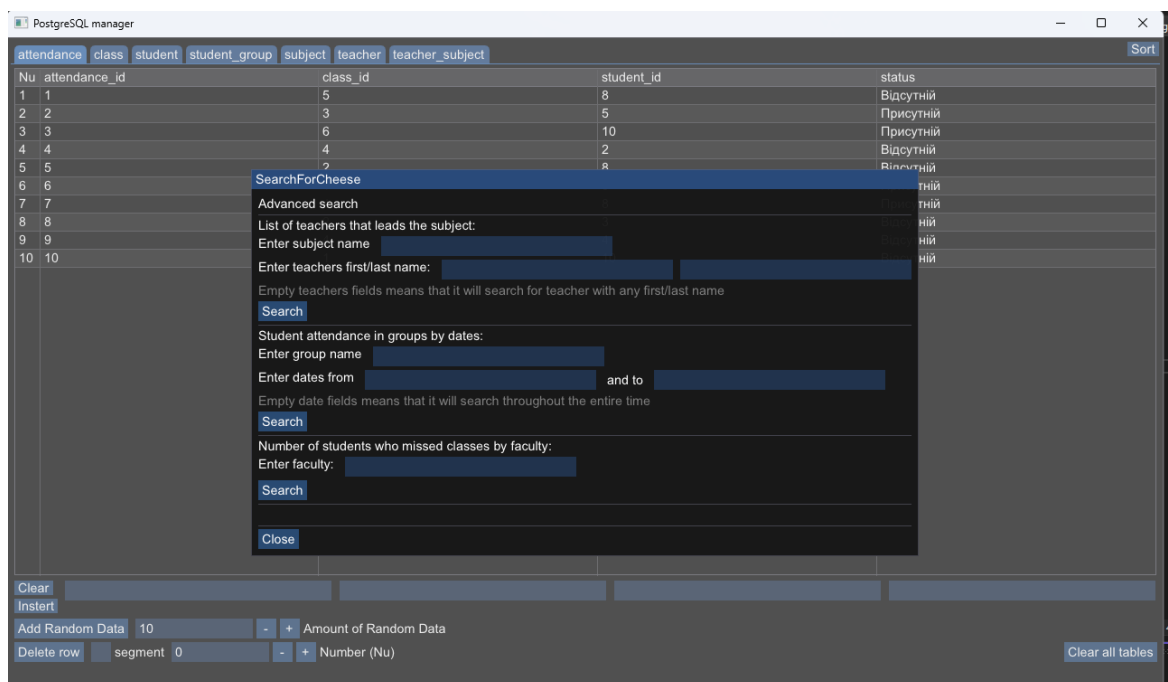


Рисунок 5 – Меню пошуку з заготовленими запитами

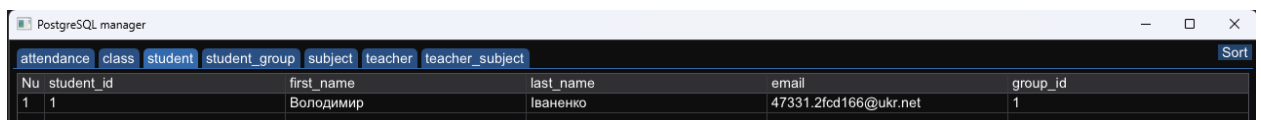
## Опис функціональності пунктів меню користувача

На рис. 4 видно:

- Вкладки – кожна з них є клікабельною та відкриває таблицю, яка відображає вміст такої ж таблиці у базі даних.
- Поля під таблицею – ввід значень з клавіатури для вставлення нового рядку в таблицю.
- Кнопки *Clear* та *Insert* – очищують поля ручного вводу.
- Кнопка *Add Random Data* – генерує запит на вставку випадкових даних в кількості, вказаній на наступному полі введення.
- Кнопка *Delete Row* – видаляє рядок з вказаним на полі введення номером. Якщо стоїть галочка *segment*, видаляє проміжок.

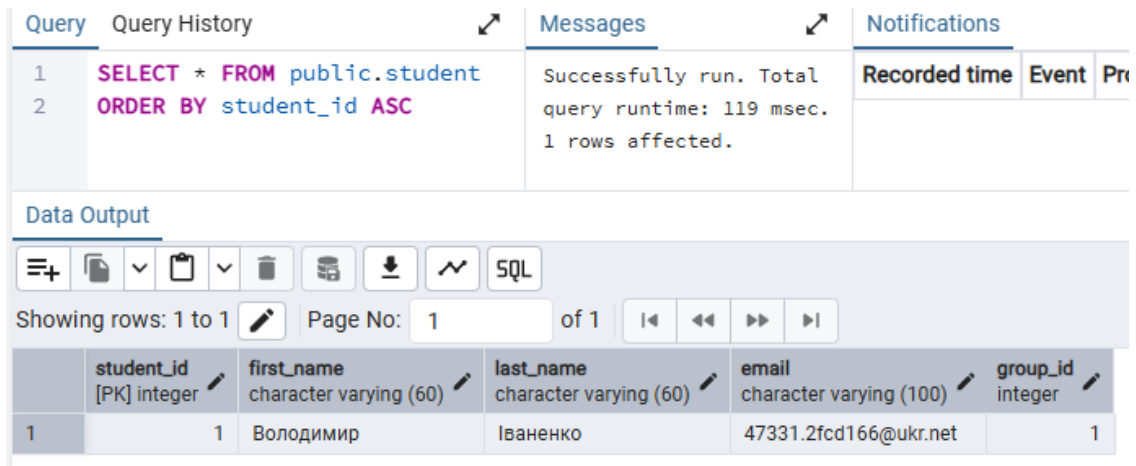
## Тестування роботи функцій програми в різних ситуаціях

Запис до оновлення:



attendance	class	student	student_group	subject	teacher	teacher_subject
Nu	student_id	first_name	last_name	email	group_id	
1	1	Володимир	Іваненко	47331.2fcd166@ukr.net	1	

Вміст, показаний в нашій програмі



Query Query History Messages Notifications

1 SELECT \* FROM public.student  
2 ORDER BY student\_id ASC

Successfully run. Total query runtime: 119 msec. 1 rows affected.

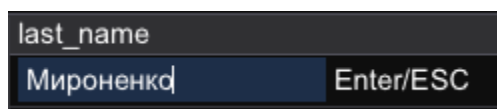
Recorded time Event Pr

Data Output

Showing rows: 1 to 1 Page No: 1 of 1

student_id [PK] integer	first_name character varying (60)	last_name character varying (60)	email character varying (100)	group_id integer
1	Володимир	Іваненко	47331.2fcd166@ukr.net	1

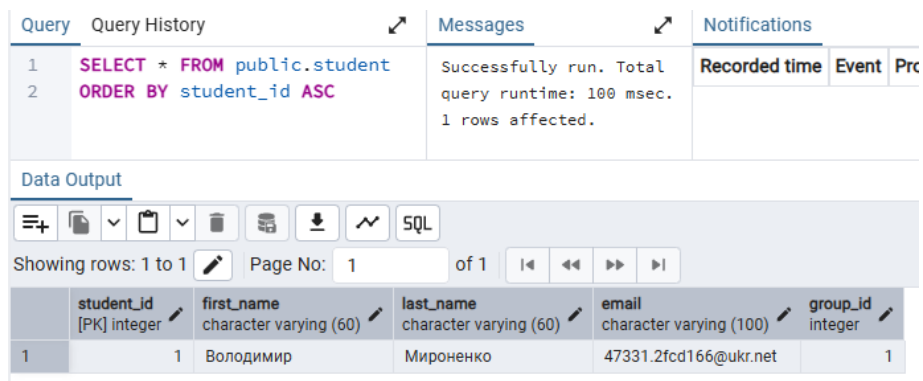
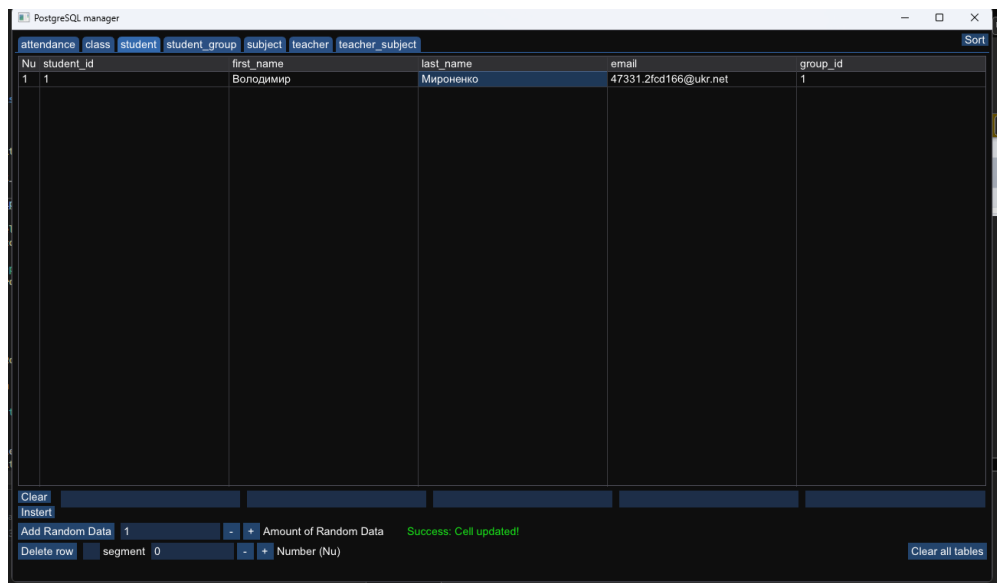
Вміст через pgAdmin4



last_name
Мироненко Enter/ESC

Редагуємо вміст комірки

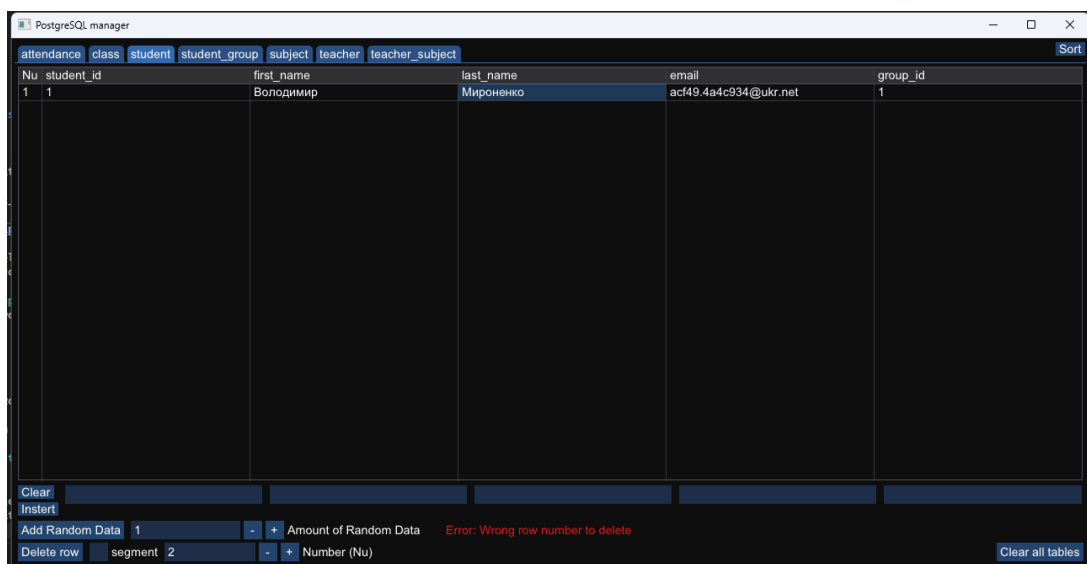
## Оновлення:



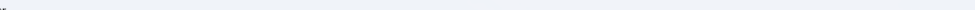
*Перевірка у pgAdmin*

## Вилучення

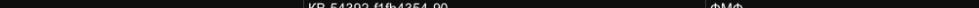
При спробі вилучити неіснуючого рядку з таблиці, або з неіснуючої таблиці програма виведе користувачу помилку про виконання такого запиту.



При спробі вилучити запис, на який міститься посилання в будь якій іншій таблиці буде видалено усі записи, які містять посилання на цей.




attendance	class	student	student_group	subject	teacher	teacher_subject	group_id
1	1	Володимир			Мироненко		1
2	3	Петро			Бойко		1



The screenshot shows the PostgreSQL manager interface. At the top, there's a title bar "PostgreSQL manager" with standard window controls. Below it, a navigation bar contains tabs: "attendance", "class", "student", "student\_group", "subject", "teacher", "teacher\_subject", and "Sort". The "attendance" tab is selected. The main area displays a table with the following structure:

Nu	group_id	group_name	faculty
1	1	KB-54392-f1fb4354-90	ΦΜΦ

The screenshot shows the PostgreSQL manager interface. At the top, there are tabs for different tables: attendance, class, student, student\_group, subject, teacher, and teacher\_subject. The 'student\_group' tab is selected. Below the tabs, there is a table with the following columns: Nu, group\_id, group\_name, and faculty. The table is empty. Below the table, there are buttons for 'Clear', 'Insert', 'Add Random Data', and 'Delete row'. The 'Delete row' button is highlighted, and a message 'Success: Rows deleted!' is displayed. At the bottom right, there is a button 'Clear all tables'.



The screenshot shows the PostgreSQL manager window. At the top, there's a toolbar with buttons for 'attendance', 'class', 'student', 'student\_group', 'subject', 'teacher', and 'teacher\_subject'. The 'student' button is selected. Below the toolbar, there's a table with the following columns: 'student\_id', 'first\_name', 'last\_name', 'email', and 'group\_id'. The table is currently empty.

Query

Query History

↗

Messages

↗

Notifications

Recorded time

Event

Progress

1

SELECT \* FROM public.student

2

ORDER BY student\_id ASC

Successfully run. Total query runtime: 60 msec. 0 rows affected.

Data Output

≡+

📄

▼

📋

▼

🗑️

📦

⬇️

📈

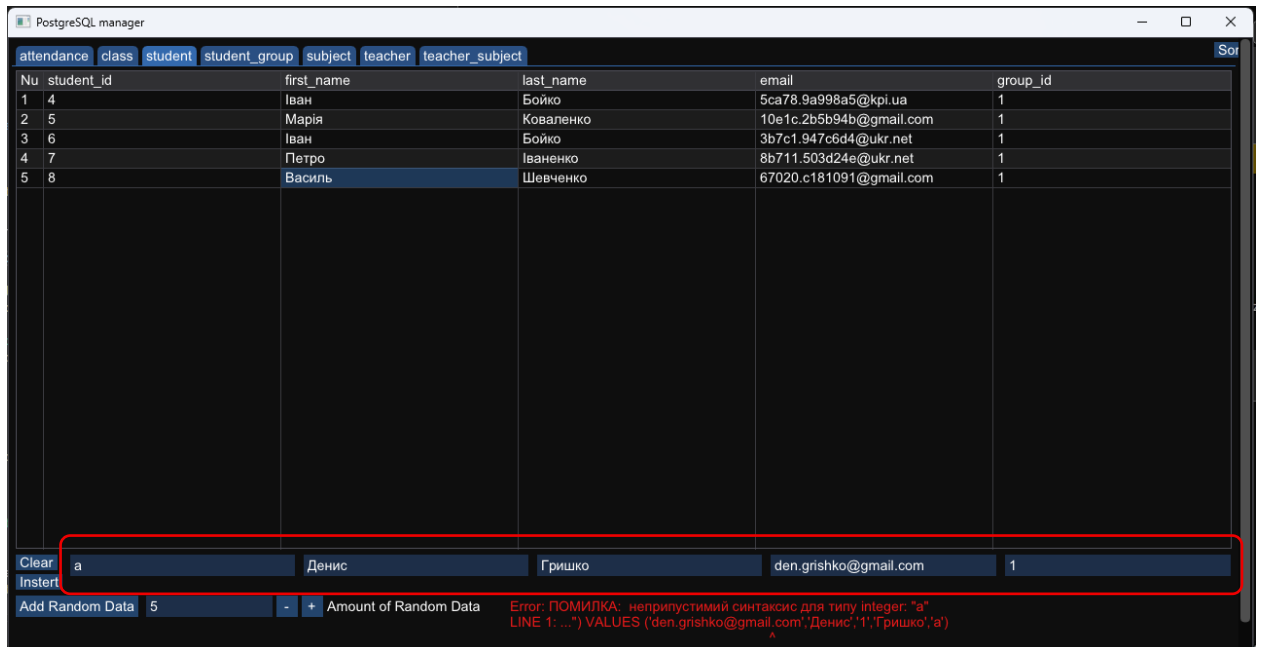
SQL

	student_id [PK] integer	first_name character varying (60)	last_name character varying (60)	email character varying (100)	group_id integer

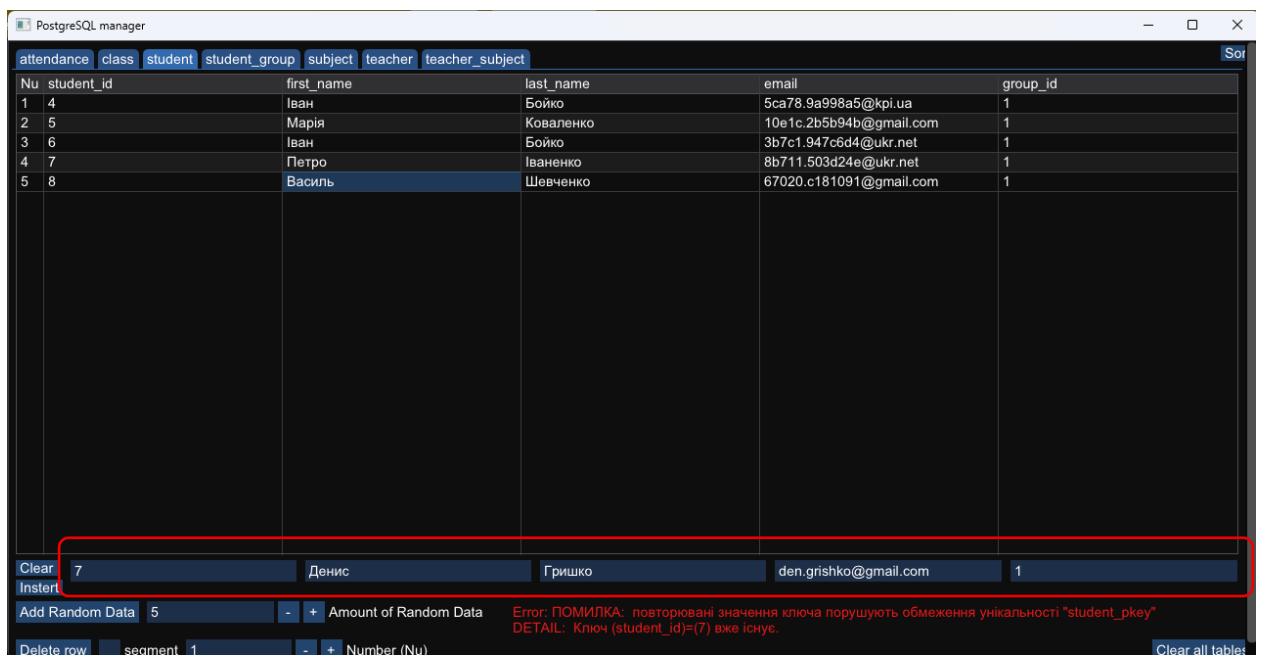
## Створення

Помилки перхоплюються, виводяться користувачу, програма повертається до меню і продовжує роботу, а до таблиць змін ніяких не внесено

Використання неправильного типу:



Використання неправильного ключа:



## Пусті інпути:

PostgreSQL manager

attendance class student student\_group subject teacher teacher\_subject Sort

Nu	student_id	first_name	last_name	email	group_id
1	4	Іван	Бойко	5ca78.9a998a5@kpi.ua	1
2	5	Марія	Коваленко	10e1c.2b5b94b@gmail.com	1
3	6	Іван	Бойко	3b7c1.947c6d4@ukr.net	1
4	7	Петро	Іваненко	8b711.503d24e@ukr.net	1
5	8	Василь	Шевченко	67020.c181091@gmail.com	1

Clear

Insert

Add Random Data 5 - + Amount of Random Data Error: No data to insert. All fields are empty.

Delete row segment 1 - + Number (Nu) Clear all tables

PostgreSQL manager

attendance class student student\_group subject teacher teacher\_subject Sort

Nu	student_id	first_name	last_name	email	group_id
1	4	Іван	Бойко	5ca78.9a998a5@kpi.ua	1
2	5	Марія	Коваленко	10e1c.2b5b94b@gmail.com	1
3	6	Іван	Бойко	3b7c1.947c6d4@ukr.net	1
4	7	Петро	Іваненко	8b711.503d24e@ukr.net	1
5	8	Василь	Шевченко	67020.c181091@gmail.com	1

Clear

Insert

Add Random Data 5 - + Amount of Random Data Error: ПОМИЛКА: null значення в стовпці "email" відношення "student" порушує not-null обмеження  
DETAIL: Помилковий рядок містить (11, n, p, null, null).

Delete row segment 1 - + Number (Nu) Clear all tables

## Використання неіснуючого зовнішнього ключа:

PostgreSQL manager

attendance class student student\_group subject teacher teacher\_subject Sort

Nu	student_id	first_name	last_name	email	group_id
1	4	Іван	Бойко	5ca78.9a998a5@kpi.ua	1
2	5	Марія	Коваленко	10e1c.2b5b94b@gmail.com	1
3	6	Іван	Бойко	3b7c1.947c6d4@ukr.net	1
4	7	Петро	Іваненко	8b711.503d24e@ukr.net	1
5	8	Василь	Шевченко	67020.c181091@gmail.com	1

Clear

Insert

Add Random Data 5 - + Amount of Random Data Error: ПОМИЛКА: insert або update в таблиці "student" порушує обмеження зовнішнього ключа "student\_group\_id"  
DETAIL: Ключ (group\_id)=(2) не присутній в таблиці "student\_group".

Delete row segment 1 - + Number (Nu) Clear all tables

## Додавання рядку з коректним id групи:

PostgreSQL manager

attendance class student student\_group subject teacher teacher\_subject Sort

Nu	student_id	first_name	last_name	email	group_id
1	4	Іван	Бойко	5ca78.9a998a5@kpi.ua	1
2	5	Марія	Коваленко	10e1c.2b5b94b@gmail.com	1
3	6	Іван	Бойко	3b7c1.947c6d4@ukr.net	1
4	7	Петро	Іваненко	8b711.503d24e@ukr.net	1
5	8	Василь	Шевченко	67020.c181091@gmail.com	1
6	1	Денис	Гришко	something@gmail.com	1

Clear Insert

Add Random Data 5 - + Amount of Random Data Success: Row inserted successfully!

Delete row segment 1 - + Number (Nu) Clear all tables

## Генерація

PostgreSQL manager

attendance class student student\_group subject teacher teacher\_subject Sort

Nu	teacher_id	first_name	last_name
----	------------	------------	-----------

Add Random Data 100 - + Amount of Random Data

PostgreSQL manager

attendance class student student\_group subject teacher teacher\_subject Sort

Nu	teacher_id	first_name	last_name
1	1	Анна	Петренко
2	2	Микита	Шевченко
3	3	Микита	Петренко
4	4	Ярослав	Бойко
5	5	Микита	Боженко
6	6	Ярослав	Сектим
7	7	Ганна	Іваненко
8	8	Анна	Петренко
9	9	Михайло	Мироненко
10	10	Михайло	Сектим
11	11	Петро	Мельник
12	12	Артем	Сектим
13	13	Анастасія	Мельник
14	14	Володимир	Бойко
15	15	Ярослав	Петренко
16	16	Вікторія	Файнберг
17	17	Денис	Мироненко
18	18	Михайло	Мироненко
19	19	Денис	Боженко
20	20	Руслан	Боженко
21	21	Марія	Бойко
22	22	Михайло	Шевченко
23	23	Марія	Шевченко
24	24	Іван	Файнберг
25	25	Марія	Гончаренко
26	26	Петро	Бойко
27	27	Микита	Коваленко

Clear Insert

Add Random Data 100 - + Amount of Random Data Success: Random data added!

Delete row segment 0 - + Number (Nu) Clear all tables

Lab1/postgres@Lab1

Query Query History Messages Notifications

```
1 SELECT * FROM public.teacher
2 ORDER BY teacher_id ASC
```

Successfully run. Total query runtime: 61 msec. 100 rows affected.

Recorded time Event Process ID Pay

Data Output

Showing rows: 1 to 100 Page No: 1 of 1

	teacher_id [PK] integer	first_name character varying (60)	last_name character varying (60)
1	1	Анна	Петренко
2	2	Микита	Шевченко
3	3	Микита	Петренко
4	4	Ярослав	Бойко
5	5	Микита	Боженко
6	6	Ярослав	Сектим
7	7	Ганна	Іваненко
8	8	Анна	Петренко
9	9	Михайло	Мироненко
10	10	Михайло	Сектим
11	11	Петро	Мельник
12	12	Артем	Сектим
13	13	Анастасія	Мельник
14	14	Володимир	Бойко

Total rows: 100 Query complete 00:00:00.061 CRLF Ln 2, Col 17

Великі запити:

PostgreSQL manager

attendance class student student\_group subject teacher teacher\_subject Sort

Nu	student_id	first_name	last_name	email	group_id
1	1	Василь	Гончаренко	a0b63.648cf36@ukr.net	9
2	2	Михайло	Петренко	54742.e3e52ed@gmail.com	6
3	3	Ярослав	Іваненко	2cd6d.19b4627@ukr.net	10
4	4	Іван	Мельник	6d693.1cb20ec@gmail.com	7
5	5	Руслан	Файнберг	9d62e.93c2531@kpi.ua	7
6	6	Михайло	Коваленко	beec9.b5ccc18@ukr.net	6
7	7	Василь	Одарченко	8196d.c99cef1@kpi.ua	7
8	8	Петро	Порошенко	aef74.b277e37@kpi.ua	7
9	9	Ганна	Файнберг	55008.fd7bc25@gmail.com	7
10	10	Вікторія	Гончаренко	f9c1e.6bbb11d@gmail.com	9
11	11	Вікторія	Бойко	0c2bc.94c0d60@gmail.com	6
12	12	Марія	Шевченко	f3fa6.0064c23@kpi.ua	10
13	13	Анна	Винник	fdcc2.74c1ea3@kpi.ua	10

Поточний вміст Student

Query Query History Messages Notifications

```
1 SELECT * FROM public.student
2 ORDER BY student_id ASC
```

Successfully run. Total query runtime: 59 msec. 13 rows affected.

Recorded time Event Process ID Pay

Data Output

Showing rows: 1 to 13 Page No: 1 of 1

	student_id [PK] integer	first_name character varying (60)	last_name character varying (60)	email character varying (100)	group_id integer
1	1	Василь	Гончаренко	a0b63.648cf36@ukr.net	9
2	2	Михайло	Петренко	54742.e3e52ed@gmail.co...	6
3	3	Ярослав	Іваненко	2cd6d.19b4627@ukr.net	10
4	4	Іван	Мельник	6d693.1cb20ec@gmail.co...	7
5	5	Руслан	Файнберг	9d62e.93c2531@kpi.ua	7
6	6	Михайло	Коваленко	beec9.b5ccc18@ukr.net	6
7	7	Василь	Одарченко	8196d.c99cef1@kpi.ua	7
8	8	Петро	Порошенко	aef74.b277e37@kpi.ua	7
9	9	Ганна	Файнберг	55008.fd7bc25@gmail.com	7
10	10	Вікторія	Гончаренко	f9c1e.6bbb11d@gmail.com	9
11	11	Вікторія	Бойко	0c2bc.94c0d60@gmail.com	6
12	12	Марія	Шевченко	f3fa6.0064c23@kpi.ua	10
13	13	Анна	Винник	fdcc2.74c1ea3@kpi.ua	10

✓ Successfully run. Total query runtime: 59 msec. 13 rows affected. ✕

Total rows: 13 Query complete 00:00:00.059 CRLF Ln 1, Col 29

Генеруємо 100к:

PostgreSQL manager

attendance class student student\_group subject teacher teacher\_subject

Sort

Nu	student_id	first_name	last_name	email	group_id
1	1	Василь	Гончаренко	a0b63.648cf36@ukr.net	9
2	2	Михайло	Петренко	54742.e3e52ed@gmail.com	6
3	3	Ярослав	Іваненко	2cd6d.19b4627@ukr.net	10
4	4	Іван	Мельник	6d693.1cb20ec@gmail.com	7
5	5	Руслан	Файнберг	9d62e.93c2531@kpi.ua	7
6	6	Михайло	Коваленко	beec9.b5ccc18@ukr.net	6
7	7	Василь	Одарченко	8196d.c99cef1@kpi.ua	7
8	8	Петро	Порошенко	aef74.b277e37@kpi.ua	7
9	9	Ганна	Файнберг	55008.fd7bc25@gmail.com	7
10	10	Вікторія	Гончаренко	f9c1e.6bbb11d@gmail.com	9
11	11	Вікторія	Бойко	0c2bc.94c0d60@gmail.com	6
12	12	Марія	Шевченко	f3fa6.0064c23@kpi.ua	10
13	13	Анна	Винник	fdcc2.74c1ea3@kpi.ua	10
14	14	Вікторія	Одарченко	d2c2f.8beed9f@gmail.com	10
15	15	Ганна	Файнберг	bdab0.c4524ad@kpi.ua	9
16	16	Микита	Мироненко	a7c0b.19d5faf@gmail.com	10
17	17	Артем	Боженко	c213e.385abf1@ukr.net	9
18	18	Артем	Петренко	e4363.ea31ba7@ukr.net	7
19	19	Ганна	Гончаренко	fa6b1.b681dd9@gmail.com	9
20	20	Артем	Петренко	0d7e2.ba60b42@gmail.com	8
21	21	Ганна	Шевченко	f8378.d90f10d@ukr.net	6
22	22	Анастасія	Порошенко	bc263.f2034df@kpi.ua	10
23	23	Марія	Бойко	43433.d51369a@ukr.net	6
24	24	Марія	Коваленко	59c0f.dc100b5@gmail.com	7
25	25	Марія	Сектим	cec81.474abeb@ukr.net	10
26	26	Іван	Коваленко	80a09.ca1cd37@kpi.ua	10
27	27	Володимир	Іваненко	92e8e.f180691@ukr.net	9

Clear

Insert

Add Random Data 100000 - + Amount of Random Data Success: Random data added!

Delete row segment 4 - + Number (Nu)

Clear all tables

Lab1/postgres@Lab1

Query Query History Messages Notifications

1 SELECT \* FROM public.student

2 ORDER BY student\_id ASC

Successfully run. Total query runtime: 116 msec. 100013 rows affected.

Recorded time Event Process ID Pay

Data Output

Showing rows: 1 to 1000 Page No: 1 of 101

	student_id [PK] integer	first_name character varying (60)	last_name character varying (60)	email character varying (100)	group_id integer
1	1	Василь	Гончаренко	a0b63.648cf36@ukr.net	9
2	2	Михайло	Петренко	54742.e3e52ed@gmail.co...	6
3	3	Ярослав	Іваненко	2cd6d.19b4627@ukr.net	10
4	4	Іван	Мельник	6d693.1cb20ec@gmail.co...	7
5	5	Руслан	Файнберг	9d62e.93c2531@kpi.ua	7
6	6	Михайло	Коваленко	beec9.b5ccc18@ukr.net	6
7	7	Василь	Одарченко	8196d.c99cef1@kpi.ua	7
8	8	Петро	Порошенко	aef74.b277e37@kpi.ua	7
9	9	Ганна	Файнберг	55008.fd7bc25@gmail.co...	7
10	10	Вікторія	Гончаренко	f9c1e.6bbb11d@gmail.co...	9
11	11	Вікторія	Бойко	0c2bc.94c0d60@gmail.co...	6
12	12	Марія	Шевченко	f3fa6.0064c23@kpi.ua	10
13	13	Анна			
14	14	Вікторія			

Successfully run. Total query runtime: 116 msec. 100013 rows affected.

Total rows: 100013 Query complete 00:00:00.116 CRLF Ln 1, Col 29

PgAdmin повідомляє що тепер в таблиці знаходиться 1000013 записів. Ідентифікатори груп, на яких посилаються згенеровані записи обираються випадковим чином серед існуючих. Інші дані ( крім id) теж обираються випадковим чином.

## Пошук

Перевірка пошуку списку викладачів які ведуть предмет:

Search for Cheese

Advanced search

List of teachers that leads the subject:  
Enter subject name KC  
Enter teachers first/last name:  
Empty teachers fields means that it will search for teacher with any first/last name  
Search

Student attendance in groups by dates:  
Enter group name  
Enter dates from and to  
Empty date fields means that it will search throughout the entire time  
Search

Number of students who missed classes by faculty:  
Enter faculty:  
Search

first_name	last_name	subject_name
Анна	Мельник	КС
Ганна	Мироненко	КС

Close

List of teachers that leads the subject:  
Enter subject name КГ  
Enter teachers first/last name:  
Empty teachers fields means that it will search for teacher with any first/last name  
Search

first_name	last_name	subject_name
Микита	Бойко	КГ
Ганна	Бойко	КГ
Ярослав	Одарченко	КГ

Advanced search

List of teachers that leads the subject:  
Enter subject name К%  
Enter teachers first/last name:  
Empty teachers fields means that it will search for teacher with any first/last name  
Search

first_name	last_name	subject_name
Ганна	Бойко	КГ
Микита	Бойко	КГ
Ярослав	Одарченко	КГ
Анна	Мельник	КС
Ганна	Мироненко	КС

Пошук також за іменем викладача

List of teachers that leads the subject:

Enter subject name К%

Enter teachers first/last name: Г%

Empty teachers fields means that it will search for teacher with any first/last name

Search

Student attendance in groups by dates:

first_name	last_name	subject_name
Ганна	Бойко	КГ
Ганна	Мироненко	КС

Отримання усіх значень

Advanced search

List of teachers that leads the subject:

Enter subject name %

Enter teachers first/last name:

Empty teachers fields means that it will search for teacher with any first/last name

Search

Student attendance in groups by dates:

first_name	last_name	subject_name
Михайло	Боженко	АМО
Ганна	Бойко	АМО
Ярослав	Бойко	АМО
Марія	Мироненко	АМО
Анна	Порошенко	АМО
Микита	Бойко	КГ
Ганна	Бойко	КГ
Ярослав	Одарченко	КГ
Анна	Мельник	КС
Ганна	Мироненко	КС

Вміст таблиць, серед яких проводився пошук:

attendance	class	student	student_group	subject	teacher	teacher_subject	Sort
Nu	subject_id			subject_name			
1	1			КС			
2	2			КГ			
3	3			АМО			

	attendance	class	student	student_group	subject	teacher	teacher_subject	Sort
Nu	teacher_id		first_name				last_name	
1	1		Микита				Мироненко	
2	2		Михайло				Файнберг	
3	3		Василь				Бойко	
4	4		Василь				Іваненко	
5	5		Артем				Боженко	
6	6		Ганна				Мироненко	
7	7		Марія				Мироненко	
8	8		Василь				Іваненко	
9	9		Артем				Петренко	
10	10		Анна				Боженко	
11	11		Анна				Порошенко	
12	12		Анна				Мельник	
13	13		Михайло				Гончаренко	
14	14		Микита				Коваленко	
15	15		Василь				Винник	
16	16		Ярослав				Бойко	
17	17		Михайло				Боженко	
18	18		Василь				Бойко	
19	19		Анна				Мироненко	
20	20		Ганна				Бойко	
21	21		Анастасія				Файнберг	
22	22		Василь				Сектим	
23	23		Денис				Гончаренко	
24	24		Ярослав				Одарченко	
25	25		Микита				Бойко	

Nu	teacher_id	subject_id
1	25	2
2	20	3
3	7	3
4	16	3
5	17	3
6	11	3
7	12	1
8	20	2
9	6	1
10	24	2

Пошук кількості відвідувань студентів за певний проміжок часу по групам

Student attendance in groups by dates:

Enter group name

Enter dates from  and to

Empty date fields means that it will search throughout the entire time

first_name	last_name	group_name	present_count
Володимир	Порошенко	KB-32	5
Микита	Сектим	KB-32	3
Анастасія	Іваненко	KB-32	2
Марія	Петренко	KB-32	2
Михайло	Сектим	KB-32	2
Іван	Шевченко	KB-32	2
Марія	Мельник	KB-32	1
Вікторія	Мироненко	KB-33	1
Анна	Одарченко	KB-33	0

Success: search time in ms 3

## Пошук кількості пропусків по факультетам

Number of students who missed classes by faculty:

Enter faculty:

faculty	absent_student_count
ФМФ	140
ФПМ	129
ФІОТ	125

Number of students who missed classes by faculty:

Enter faculty:

faculty	absent_student_count
ФПМ	129

### Фрагменти коду:

Нижче наведені деякі приклади що ілюструють функціональні можливості додатку.

### Перегляд даних

```
const pqxx::result& Model::requestTableData(const std::string& TableName) {  
    if (!conn)  
        return pqxx::result{};  
    try {  
        pqxx::work worker(*conn);  
        pqxx::result res = worker.exec("SELECT * FROM " + worker.quote_name(TableName));  
        tableData = { TableName, res };  
        return tableData.second;  
    }  
    catch (const std::exception& e) {  
        throw std::runtime_error(e.what());  
        return pqxx::result{};  
    }  
}
```

Ця функція requestTableData потрібна для отримання даних вказаної таблиці.

**Параметри:** TableName — рядок, який містить назву таблиці, дані якої ми будемо отримувати.

**Запит:** Вибирає всі записи з таблиці TableName;

**Результат:** Повертає посилання на “сирі” дані, з яких можна читати.

**Обробка помилок:** У разі помилки повертає повідомлення, яке надає нам бд.

## Створення і внесення даних

```
void Model::insertData(const std::string& tableName, const std::map<std::string, std::string>& data) {
    if (data.empty())
        throw std::runtime_error("No data to insert");

    try {
        pqxx::work worker(*conn);
        std::string request = "INSERT INTO " + worker.quote_name(tableName) + " (";
        std::string cols = "", vals = "";
        for (const auto& pair : data) {
            if (pair.first.find("status") != std::string::npos && tableName.find("attendance") != std::string::npos) {
                if (pair.second != "Присутній" && pair.second != "Відсутній") {
                    throw std::runtime_error("Wrong status! Should be 'Присутній' or 'Відсутній'");
                }
            }
            cols += worker.quote_name(pair.first) + ",";
            vals += worker.quote(pair.second) + ",";
        }
        cols.pop_back();
        vals.pop_back();
        request += cols + ") VALUES (" + vals + ")";

        worker.exec(request);
        worker.commit();
        requestTableData(tableName);
    }
    catch (const std::exception& e) {
        throw std::runtime_error(e.what());
    }
}
```

Ця функція insertData додає новий запис у таблицю tableName.

**Параметри:** tableName, data— назва таблиці та дані для вставки ( колонка – вміст).

**Запит:** Вставляє новий рядок у вказану таблицю де вказані колонки і дані, які в них вставити.

**Результат:** Оновлює існуючі збережені дані таблиці після вставки нових.

**Обробка помилок:** У разі помилки скасовує транзакцію і виводить повідомлення про помилку. Також перевіряє чи є значення статусу відвідування в рядку і його коректність.

## Оновлення даних

```
void Model::updateDataCell(const std::string& tableName, pqxx::row dataRow, unsigned int
```

```

changedCellNumber, const std::string& dataToChange) {
    if (dataRow.size() <= 0) {
        throw std::runtime_error("Nothing to update?");
        return;
    }
    try {
        std::vector<std::string> colNames = getTableColumnsNames(tableName);

        if (colNames[changedCellNumber].find("status") != std::string::npos &&
            tableName.find("attendance") != std::string::npos) {
            if (dataToChange != "Присутній" && dataToChange != "Відсутній") {
                throw std::runtime_error("Wrong status! Should be 'Присутній' or 'Відсутній'");
            }
        }

        pqxx::work worker(*conn);
        std::string request = "UPDATE " + worker.quote_name(tableName) + " SET " +
            colNames[changedCellNumber] + " = " + worker.quote(dataToChange) + " WHERE ";
        unsigned int i = 0;
        for (const auto& element : dataRow) {
            if (i != changedCellNumber) {
                request += worker.quote_name(colNames[i]);
                if (element.is_null()) {
                    request += " IS NULL AND ";
                }
                else {
                    request += " = " + worker.quote(element.as<std::string>()) + " AND ";
                }
            }
            i++;
        }
        if (request.size() >= 5) request.resize(request.size() - 5);

        worker.exec(request);
        worker.commit();
        requestTableData(tableName);
    }
    catch (const std::exception& e) {
        throw std::runtime_error(e.what());
    }
}

```

Ця функція `updateDataCell` оновлює інформацію у вказаній таблиці, у вказаній клітинці

### Параметри:

- `tableName` — назва таблиці.
- `dataRow`, `changedCellNumber`, `dataToChange` — рядок де буде вставлення, номер колонки де буде зміна та дані, які замінити.

**Запит:** Оновлює запис у таблиці. Знаходить таблицю, потім знаходить рядок із параметрів, отримує список колонок і знаючи номер колонки дізнається куди саме вставляти нові дані.

**Результат:** Оновлює дані в конкретній клітинці, потім оновлює дані збережені дані таблиці.

**Обробка помилок:** У разі помилки скасовує транзакцію і виводить повідомлення про помилку. Також перевіряє чи є це колонка status та перевіряє чи коректне значення статусу вводиться.

## Пошук

```
pqxx::result Model::executeCustomQuery(const std::string& request, pqxx::params params) {
    try {
        pqxx::work worker(*conn);
        return worker.exec(request, params);
    }
    catch (const std::exception& e) {
        throw std::runtime_error(e.what());
    }
}
```

Ця функція executeCutomQuery дозволяє виконати запит до бд з заданими параметрами, повертаючи результати цього запиту. Самі ж запити пошуку є заготовленими для тестів і знаходяться у контролері:

```
void Controller::on1SearchRequestClicked() {

    if (strlen(getSubjectBuffer()) <= 0) {
        response.commitErrorMessage("no subject name");
        return;
    }

    std::string request = "SELECT teacher.first_name, teacher.last_name, subject.subject_name"
        " FROM teacher JOIN teacher_subject ON teacher.teacher_id = teacher_subject.teacher_id "
        "JOIN Subject ON teacher_subject.subject_id = subject.subject_id "
        "WHERE  subject.subject_name LIKE $1 AND teacher.first_name LIKE $2 AND teacher.last_name"
    LIKE $3 "
        "ORDER BY subject.subject_name, teacher.last_name";

    std::string subject = getSubjectBuffer();
    std::string first_name = (strlen(getFirstNameBuffer()) > 0) ? getFirstNameBuffer() : "";
    std::string last_name = (strlen(getLastNameBuffer()) > 0) ? getLastNameBuffer() : "";

    try {
        auto start = std::chrono::high_resolution_clock::now();
        search_res = model.executeCustomQuery(request, { subject, first_name, last_name });
        auto end = std::chrono::high_resolution_clock::now();
        response.commitSuccessMessage(std::string(search_res.size() > 0 ? "" : "nothing found, ") + " search"
time in ms "
        + std::to_string(duration_cast<std::chrono::milliseconds>(end - start).count()));
        custom_quert_buffers["Subject"].fill('\0');
        custom_quert_buffers["FirstName"].fill('\0');
        custom_quert_buffers["LastName"].fill('\0');
    }
    catch (const std::exception& e) {
        response.commitErrorMessage(e.what());
    }
}
```

У свою чергу ця функція on1SearchRequestClicked() є заготовленим запитом, який отримує значення subject, first\_name та last\_name й шукає усіх викладачів, які належать до певного предмету. Як результат виводить імя,

прізвище та предмет, до якого належить викладач. Також виводить час, витрачений на запит.

## Видалення даних

```
void Model::deleteDataRange(const std::string& tableName, const pqxx::result& dataToDelete, int rowFrom, int rowTo)
{
    if (rowFrom > rowTo) {
        std::swap(rowFrom, rowTo);
    }
    if (dataToDelete.empty()) {
        throw std::runtime_error("No data to delete");
    }

    std::string tempTableName = "temp_pks_to_delete_" + std::to_string(std::time(nullptr));

    try {
        pqxx::work worker(*conn);

        std::map<std::string, std::string> pkInfo = getPrimaryKeyInfo(worker, tableName);
        if (pkInfo.empty()) {
            throw std::runtime_error("No PK in table");
        }

        std::vector<int> pkIndicesInResult;
        std::string pkColsForCreate;

        std::vector<std::string> pkColNamesForCopy;

        std::string pkColsForJoin;

        try {
            for (const auto& pair : pkInfo) {
                const std::string& colName = pair.first;
                const std::string& colType = pair.second;

                pkIndicesInResult.push_back(dataToDelete.column_number(colName));
                std::string quotedName = worker.quote_name(colName);

                pkColsForCreate += quotedName + " " + colType + ",";
                pkColNamesForCopy.push_back(colName);
                pkColsForJoin += "t." + quotedName + " = tmp." + quotedName + " AND ";
            }
        }
        catch (const std::exception& e) {
            throw std::runtime_error("Can not find PK column: " + std::string(e.what()));
        }

        pkColsForCreate.pop_back();
        pkColsForJoin.resize(pkColsForJoin.size() - 5);

        std::string createTempTableSql = "CREATE TEMPORARY TABLE " +
            worker.quote_name(tempTableName) +
            " (" + pkColsForCreate + ") ON COMMIT DROP;";
        worker.exec(createTempTableSql);

        std::string full = tempTableName + "(";
        for (size_t i = 0; i < pkColNamesForCopy.size(); ++i)
```

```

{
    full += worker.quote_name(pkColNamesForCopy[i]);
    if (i + 1 != pkColNamesForCopy.size()) full += ",";
}
full += ")";

auto copier = pqxx::stream_to::raw_table(worker, full);

for (int i = rowFrom; i <= rowTo; ++i) {
    const auto& row = dataToDelete[i];
    std::vector<std::string> values;
    for (size_t j = 0; j < pkIndicesInResult.size(); ++j) {
        const auto& field = row[pkIndicesInResult[j]];
        if (field.is_null()) {
            values.push_back("\\N");
        }
        else {
            values.push_back(field.c_str());
        }
    }
    copier.write_values(values);
}
copier.complete();

std::string deleteSql = "DELETE FROM " + worker.quote_name(tableName) + " t "
    "USING " + worker.quote_name(tempTableName) + " tmp "
    "WHERE " + pkColsForJoin;
worker.exec(deleteSql);

worker.commit();
requestTableData(tableName);
}
catch (const std::exception& e) {
    throw std::runtime_error("Deleting data error: " + std::string(e.what()));
}
}

```

Ця функція `deleteDataRange` видаляє записи з таблиці `tableName` на проміжку `rowFrom` та `rowTo`.

### Параметри:

- `tableName` — назва таблиці, з якої видаляються дані.
- `dataToDelete` — дані, рядки з яких треба видалити.
- `rowFrom`, `rowTo` — номери рядків, які позначають межі видалення.

**Запит:** Видаляє всі записи з `tableName`, де `pk` перетинаються з тимчасовою таблицею.

**Результат:** Оновлюємо значення таблиці, які зберігаються локально.

**Обробка помилок:** У разі помилки скасовує транзакцію і виводить повідомлення про помилку.

## Генерація даних

```
void Model::generateRandomDataForTable(const std::string& tableName, int rowsCount) {
    try {
        if (!conn)
            reconnect();

        pqxx::work txn(*conn);

        if (tableName.find("teacher_subject") != std::string::npos)
        {
            // Перевіряємо, чи є вхідні дані
            pqxx::result t_count = txn.exec("SELECT COUNT(*) FROM Teacher;");
            pqxx::result s_count = txn.exec("SELECT COUNT(*) FROM Subject;");
            if (t_count[0][0].as<int>() == 0 || s_count[0][0].as<int>() == 0) {
                throw std::runtime_error("cannot generate links - Teacher or Subject table is empty.");
            }

            std::string insertSQL =
                "INSERT INTO teacher_subject (teacher_id, subject_id) "
                "SELECT t.teacher_id, s.subject_id "
                "FROM Teacher t "
                "CROSS JOIN Subject s "
                "LEFT JOIN teacher_subject ts "
                " ON t.teacher_id = ts.teacher_id AND s.subject_id = ts.subject_id "
                "WHERE ts.teacher_id IS NULL "
                "ORDER BY random() "
                "LIMIT $1";

            txn.exec(insertSQL, rowsCount);
            txn.commit();
            requestTableData(tableName);
            return;
        }

        pqxx::result fks = txn.exec("(SELECT kcu.column_name, ccu.table_name AS foreign_table, ccu.column_name
AS foreign_column "
            "FROM information_schema.table_constraints AS tc JOIN information_schema.key_column_usage AS kcu
ON tc.constraint_name = kcu.constraint_name "
            "JOIN information_schema.constraint_column_usage AS ccu ON ccu.constraint_name = tc.constraint_name
"
            "WHERE tc.table_name = " + txn.quote(tableName) + " AND tc.constraint_type = 'FOREIGN KEY' AND
tc.table_schema = 'public')");

        std::vector<std::string> missingTables;
        for (auto const& row : fks) {
            std::string refTable = row["foreign_table"].c_str();
            pqxx::result count = txn.exec("SELECT COUNT(*) FROM " + txn.quote_name(refTable) + ";");
            if (count[0][0].as<int>() == 0)
                missingTables.push_back(refTable);
        }

        if (!missingTables.empty()) {
            std::string msg = "missing data in related tables: ";
            for (size_t i = 0; i < missingTables.size(); ++i) {
                msg += missingTables[i] + (i + 1 < missingTables.size() ? ", " : "");
            }
            throw std::runtime_error(msg);
        }

        pqxx::result all_cols = txn.exec("(SELECT column_name, data_type, character_maximum_length,
column_default FROM information_schema.columns "

```

```

        "WHERE table_name = " + txn.quote(tableName) + " AND table_schema = 'public' ORDER BY
ordinal_position)");

std::string colsList, valsList;
std::vector<pqxx::row> colsToInsert;

for (auto const& row : all_cols) {
    if (row["column_default"].is_null()) {
        colsToInsert.push_back(row);
    }
}

if (colsToInsert.empty()) {
    throw std::runtime_error("No columns to insert");
}

for (size_t i = 0; i < colsToInsert.size(); ++i) {
    colsList += txn.quote_name(colsToInsert[i]["column_name"].c_str());
    if (i + 1 < colsToInsert.size()) colsList += ", ";
}

for (size_t i = 0; i < colsToInsert.size(); ++i) {
    std::string colName = colsToInsert[i]["column_name"].c_str();
    std::string colType = colsToInsert[i]["data_type"].c_str();
    auto colLength = colsToInsert[i]["character_maximum_length"];
    bool isFK = false;
    std::string fkSelect;

    for (auto const& fk : fks) {
        if (fk["column_name"].c_str() == colName) {
            isFK = true;
            fkSelect = "(SELECT " + txn.quote_name(fk["foreign_column"].c_str()) +
                " FROM " + txn.quote_name(fk["foreign_table"].c_str()) +
                " ORDER BY random() LIMIT 1 OFFSET (s.i * 0))";

            break;
        }
    }

    if (isFK) {
        valsList += fkSelect;
    }
    else if (colType.find("char") != std::string::npos || colType == "text") {

        std::string generator_sql;
        bool apply_substr = true;

        if (colName.find("first_name") != std::string::npos) {
            generator_sql = reinterpret_cast<const char*>(u8"(ARRAY['Іван', 'Петро', 'Марія', 'Ярослав', 'Анна',
'Володимир', "
                u8"'Василь', 'Ганна', 'Вікторія', 'Анастасія', 'Денис', 'Артем', 'Руслан', 'Михайло',
'Микита'])(floor(random() * 15 + 1))");
        }
        else if (colName.find("last_name") != std::string::npos) {
            generator_sql = reinterpret_cast<const char*>(u8"(ARRAY['Шевченко', 'Коваленко', 'Петренко',
'Тваненко', 'Бойко', 'Мельник', "
                u8"'Одарченко', 'Порошенко', 'Мироненко', 'Сектим', 'Гончаренко', 'Винник', 'Боженко',
'Файнберг'])(floor(random() * 14 + 1))");
        }
        else if (colName.find("status") != std::string::npos) {
            generator_sql = reinterpret_cast<const char*>(u8"(ARRAY['Присутній', 'Відсутній'])(floor(random()
* 2 + 1))");

```

```

    }
    else if (colName.find("class_type") != std::string::npos) {
        generator_sql = reinterpret_cast<const char*>(u8"(ARRAY['Практика', 'Лекція',
'Лабораторна'])(floor(random() * 3 + 1))");
    }
    else if (colName.find("email") != std::string::npos) {
        generator_sql = "substr(md5(random()::text), 1, 5) || '.' || substr(md5(random()::text), 1, 7) || "
"(ARRAY['@gmail.com', '@ukr.net', '@kpi.ua'])(floor(random() * 3 + 1))";
    }
    else if (colName.find("group_name") != std::string::npos) {
        generator_sql = reinterpret_cast<const char*>(u8"(ARRAY['KB', 'KM', 'КП'])(floor(random() * 3 + 1))
|| '-' || "
        "floor(random() * 90000 + 10000)::int || '-' || gen_random_uuid()::text");
    }
    else if (colName.find("subject_name") != std::string::npos) {
        generator_sql = reinterpret_cast<const char*>(u8"(ARRAY['COHC', 'АМО', 'КЕ', 'ЗІКС', 'БДЗУ',
'КС'])(floor(random() * 6 + 1)) "
        "|| '-' || gen_random_uuid()::text");
    }
    else if (colName.find("faculty") != std::string::npos) {
        generator_sql = reinterpret_cast<const char*>(u8"(ARRAY['ФІМ', 'ФМФ', 'ФІОТ'])(floor(random() *
3 + 1))");
    }

    else {
        std::string len = "8";
        if (!colLength.is_null()) {
            int dbLen = colLength.as<int>();
            len = (std::to_string)((std::min)(dbLen, 32));
        }
        generator_sql = "substr(md5(random()::text), 1, " + len + ")";
        apply_substr = false;
    }

    if (apply_substr && !colLength.is_null()) {
        valsList += "substr(" + generator_sql + ", 1, " + colLength.as<std::string>() + ")";
    }
    else {
        valsList += generator_sql;
    }
}

else if (colType.find("int") != std::string::npos) {
    valsList += "floor(random()*1000)::int";
}
else if (colType.find("date") != std::string::npos || colType.find("timestamp") != std::string::npos) {
    valsList += "now() - (random() * interval '10 years')";
}
else if (colType == "boolean") {
    valsList += "random() > 0.5";
}
else {
    valsList += "NULL";
}

if (i + 1 < colsToInsert.size()) valsList += ", ";
}

std::string insertSQL = "INSERT INTO " + txn.quote_name(tableName) +
" (" + colsList + ") " +
" SELECT " + valsList +
" FROM generate_series(1, " + std::to_string(rowsCount) + ") AS s(i);";

```

```

        txn.exec(insertSQL.c_str()).no_rows();

        txn.commit();
        requestTableData(tableName);
    }
    catch (std::exception const& e) {
        throw std::runtime_error(e.what());
    }
}

```

Функція `generateRandomDataForTable` генерує випадкові значення для усіх таблиць. Має наперед прописані можливі варіанти назв колонок і відповідні варіанти даних для вставки під них. Ось як вона працює:

Функція автоматично генерує випадкові дані для вибраної таблиці, враховуючи всі її залежності та обмеження. Спочатку вона перевіряє підключення до бази та визначає, чи має таблиця зовнішні ключі. Якщо пов'язані таблиці порожні, генерація неможлива — у такому випадку повертається відповідна помилка. Для спеціальної таблиці `teacher_subject` використовується окремий алгоритм, який створює випадкові унікальні пари викладач–предмет, не допускаючи дублювання вже існуючих комбінацій.

Після цього функція отримує список колонок без значення за замовчуванням і формує для кожної з них власний генератор: випадкові імена, прізвища, email, назви груп, статуси, дати, числа, логічні значення або foreign key з інших таблиць. Згенеровані дані вставляються одним SQL-запитом через `generate_series`, після чого транзакція фіксується, а таблиця оновлюється в інтерфейсі. Якщо під час роботи стається помилка, вона перехоплюється і передається далі без часткових змін у базі.

Також для тестування до деяких значень додано універсальні унікальні ідентифікатори за допомогою функції `gen_random_uuid()::text`. Це потрібно для генерації даних, де є унікальні поля крім `pk`, але потрібен великий об'єм згенерованих рядків.

## Повний код програми

### main.cpp

```

#include <iostream>
#include <Windows.h>

#include "Controller.h"

int main() {
    SetConsoleOutputCP(CP_UTF8);
    SetConsoleCP(CP_UTF8);

    mvc::Controller starter;

    return EXIT_SUCCESS;
}

```

### Controller.h

```

#pragma once

#include "Model.h"
#include "r_handler.h"

```

```

namespace mvc {
    class View;

    class Controller {
        ResponseHandler response;
        View& view;
        Model& model;

        static const size_t BUFFER_SIZE = 256;

        int current_row_index = 0;

        int selected_row = -1, selected_col = -1;
        int editing_row = -1, editing_col = -1;
        char text_buffer[BUFFER_SIZE];

        int randomDataCount = 0;
        int deleteRowsFrom = 0, deleteRowsTo = 0;
        bool isSegmentToDelete = false;

        bool isSortWindowOpen = false;

        std::map<std::string, std::array<char, BUFFER_SIZE>> insert_buffers;
        std::map<std::string, std::array<char, BUFFER_SIZE>>
custom_quert_buffers;

    public:
        Controller();

        pqxx::result res, search_res;
        std::string current_table_name;

        void onCellEditConfirmed();
        void onDeleteRowsClicked();

        std::vector<std::string> getTableNames();
        pqxx::result& getTableData(const std::string& name);
        std::vector<std::string> getTableColumnsNames(const std::string&
name);

        bool isCellEditing(int r, int c);
        bool isSelectedCell(int r, int c);
        bool isSearchWindowOpen() const;

        char* getEditBuffer();

        void onCellSelected(int row, int col);
        void onCellDoubleClicked(int row, int col, const std::string&
current_text);
        void onCellEditCanceled();
        void onAddRandomDataClicked();
        void onClearAllClicked();
        void onInsertClicked();
        void onOpenSearchWindowClicked();
        void onCloseSearchWindowClicked();

        int* getRandomDataCountPtr();
        int* getDeleteRowsFromPtr();
        int* getDeleteRowsToPtr();
        bool* getIsSegmentToDeletePtr();

        void clearInsertBuffers();
        char* getInsertBuffer(const std::string& colName);

```

```

        size_t getBufferSize() const;

        void on1SearchRequestClicked();
        void on2SearchRequestClicked();
        void on3SearchRequestClicked();

        char* getSubjectBuffer();
        char* getFirstNameBuffer();
        char* getLastNameBuffer();
        char* getGroupBuffer();
        char* getAttendanceDateFromBuffer();
        char* getAttendanceDateToBuffer();
        char* getFacultyBuffer();
    };
}

```

## controller.cpp

```

#include "Controller.h"
#include "View.h"

#include <chrono>

using namespace mvc;

Controller::Controller() : view(View::Spawner::instance(response)),
model(Model::Spawner::instance())
{
    view.setController(this);
    view.Start();
}

void Controller::onCellEditConfirmed() {
    try {
        std::string s(text_buffer);

        pqxx::row row_to_update = res[editing_row];

        model.updateDataCell(current_table_name, row_to_update, editing_col,
s);
        response.commitSuccessMessage("Cell updated!");
    }
    catch (const std::exception& e) {
        response.commitErrorMessage(e.what());
    }
    editing_row = -1;
    editing_col = -1;
}

void Controller::onDeleteRowsClicked() {
    try {
        if (deleteRowsFrom <= 0 || deleteRowsFrom > res.size() ||
(isSegmentToDelete && (deleteRowsTo <= 0 || deleteRowsTo > res.size())))
        {
            throw std::runtime_error("Wrong row number to delete");
        }

        if (isSegmentToDelete) {
            model.deleteDataRange(current_table_name, res, deleteRowsFrom -
1, deleteRowsTo - 1);
        }
        else {

```

```

        model.deleteDataRange(current_table_name, res, deleteRowsFrom -
1, deleteRowsFrom - 1);
    }
    response.commitSuccessMessage("Rows deleted!");
}
catch (const std::exception& e) {
    response.commitErrorMessage(e.what());
}
}

std::vector<std::string> Controller::getTableNames() {
    return model.getTableNames();
}

pqxx::result& Controller::getTableData(const std::string& name) {
    this->res = model.getTableData(name);
    return this->res;
}

bool Controller::isCellEditing(int r, int c) {
    return editing_row == r && editing_col == c;
}

bool Controller::isCellSelected(int r, int c) {
    return selected_row == r && selected_col == c;
}

char* Controller::getEditBuffer() {
    return text_buffer;
}

void Controller::onCellSelected(int row, int col) {
    selected_row = row;
    selected_col = col;
}

void Controller::onCellDoubleClicked(int row, int col, const std::string&
current_text) {
    editing_row = row;
    editing_col = col;
    strncpy_s(text_buffer, sizeof(text_buffer), current_text.c_str(),
current_text.size());
    text_buffer[sizeof(text_buffer) - 1] = '\0';
}

void Controller::onCellEditCanceled() {
    editing_row = -1;
    editing_col = -1;
}

void Controller::onAddRandomDataClicked() {
    try {
        model.generateRandomDataForTable(current_table_name,
randomDataCount);
        response.commitSuccessMessage("Random data added! " +
std::to_string(randomDataCount));
    }
    catch (const std::exception& e) {
        response.commitErrorMessage(e.what());
    }
}

void Controller::onClearAllClicked() {
    try {
        model.clearAllTables(current_table_name);
    }
    catch (const std::exception& e) {
        response.commitErrorMessage(e.what());
    }
}

```

```

        response.commitSuccessMessage("Nothing left!");
    }
    catch (const std::exception& e) {
        response.commitErrorMessage(e.what());
    }
}

size_t Controller::getBufferSize() const {
    return BUFFER_SIZE;
}

std::vector<std::string> Controller::getTableColumnsNames(const std::string&
name) {
    return model.getTableColumnsNames(name);
}

void Controller::clearInsertBuffers()
{
    for (auto& buf : insert_buffers) {
        buf.second.fill('\0');
    }
}

char* Controller::getInsertBuffer(const std::string& colName)
{
    return insert_buffers[colName].data();
}

void Controller::onInsertClicked()
{
    try {
        std::map<std::string, std::string> dataToInsert;
        for (const auto& pair : insert_buffers)
        {
            std::string value(pair.second.data());

            if (!value.empty()) {
                dataToInsert[pair.first] = value;
            }

            if (dataToInsert.empty()) {
                throw std::runtime_error("No data to insert. All fields are
empty.");
            }

            model.insertData(current_table_name, dataToInsert);

            response.commitSuccessMessage("Row inserted successfully!");
            clearInsertBuffers();
        }
        catch (const std::exception& e)
        {
            response.commitErrorMessage(e.what());
        }
    }

void Controller::onOpenSearchWindowClicked() {
    isSortWindowOpen = true;
}

void Controller::onCloseSearchWindowClicked() {
    isSortWindowOpen = false;
}

```

```

bool Controller::isSearchWindowOpen() const {
    return isSortWindowOpen;
}

void Controller::on1SearchRequestClicked() {

    if (strlen(getSubjectBuffer()) <= 0) {
        response.commitErrorMessage("no subject name");
        return;
    }

    std::string request = "SELECT teacher.first_name, teacher.last_name,
subject.subject_name "
        " FROM teacher JOIN teacher_subject ON teacher.teacher_id =
teacher_subject.teacher_id "
        "JOIN Subject ON teacher_subject.subject_id = subject.subject_id "
        "WHERE subject.subject_name LIKE $1 AND teacher.first_name LIKE $2
AND teacher.last_name LIKE $3 "
        "ORDER BY subject.subject_name, teacher.last_name";

    std::string subject = getSubjectBuffer();
    std::string first_name = (strlen(getFirstNameBuffer()) > 0) ?
getFirstNameBuffer() : "%";
    std::string last_name = (strlen(getLastNameBuffer()) > 0) ?
getLastNameBuffer() : "%";

    try {
        auto start = std::chrono::high_resolution_clock::now();
        search_res = model.executeCustomQuery(request, { subject, first_name,
last_name });
        auto end = std::chrono::high_resolution_clock::now();
        response.commitSuccessMessage(std::string(search_res.size() > 0 ? ""
: "nothing found, ") + " search time in ms "
            + std::to_string(duration_cast<std::chrono::milliseconds>(end -
start).count()));
        custom_quert_buffers["Subject"].fill('\0');
        custom_quert_buffers["FirstName"].fill('\0');
        custom_quert_buffers["LastName"].fill('\0');
    }
    catch (const std::exception& e) {
        response.commitErrorMessage(e.what());
    }
}

void Controller::on2SearchRequestClicked() {
    if (strlen(getGroupBuffer()) <= 0) {
        response.commitErrorMessage("no group name");
        return;
    }

    std::string request = reinterpret_cast < const char*>(u8"SELECT
student.first_name, student.last_name, student_group.group_name, "
        "COUNT(attendance.attendance_id) FILTER(WHERE attendance.status =
'Присутній') AS present_count "
        "FROM Student group JOIN Student ON student_group.group_id =
student.group_id "
        "JOIN Attendance ON student.student_id = attendance.student_id JOIN
Class ON attendance.class_id = class.class_id "
        "WHERE student_group.group_name LIKE $1 AND class.class_date BETWEEN
$2 AND COALESCE($3, CURRENT_DATE) "
        "GROUP BY student.student_id, student_group.group_name "
        "ORDER BY present_count DESC, student.last_name");

```

```

        std::string group = getGroupBuffer();
        std::string dateFrom = (strlen(getAttendanceDateFromBuffer()) > 0) ?
getAttendanceDateFromBuffer() : "1970-01-01";
        std::optional<std::string> dateTo; // за замовчуванням він (std::nullopt)
        if (strlen(getAttendanceDateToBuffer()) > 0) {
            dateTo = getAttendanceDateToBuffer();
        }

        try {
            auto start = std::chrono::high_resolution_clock::now();
            search_res = model.executeCustomQuery(request, { group, dateFrom,
dateTo });
            auto end = std::chrono::high_resolution_clock::now();
            response.commitSuccessMessage(std::string(search_res.size() > 0 ? ""
: "nothing found, ") + " search time in ms "
+ std::to_string(duration_cast<std::chrono::milliseconds>(end -
start).count()));
            custom_quert_buffers["Group"].fill('\0');
            custom_quert_buffers["AttendanceDateFrom"].fill('\0');
            custom_quert_buffers["AttendanceDateTo"].fill('\0');
        }
        catch (const std::exception& e) {
            response.commitErrorMessage(e.what());
        }
    }

void Controller::on3SearchRequestClicked() {
    std::string request = reinterpret_cast< const char*>(u8"SELECT
student_group.faculty, COUNT(DISTINCT student.student_id) AS
absent_student_count "
"FROM student_group JOIN student ON student_group.group_id =
student.group_id JOIN Attendance ON student.student_id =
attendance.student_id "
"WHERE attendance.status = 'Відсутній' AND student_group.faculty LIKE
$1 "
"GROUP BY student_group.faculty ORDER BY absent_student_count DESC");

    std::string faculty = (strlen(getFacultyBuffer()) > 0) ?
getFacultyBuffer() : "%";

    try {
        auto start = std::chrono::high_resolution_clock::now();
        search_res = model.executeCustomQuery(request, { faculty });
        auto end = std::chrono::high_resolution_clock::now();
        response.commitSuccessMessage(std::string(search_res.size() > 0 ? ""
: "nothing found, ") + " search time in ms "
+ std::to_string(duration_cast<std::chrono::milliseconds>(end -
start).count()));
        custom_quert_buffers["Faculty"].fill('\0');
    }
    catch (const std::exception& e) {
        response.commitErrorMessage(e.what());
    }
}

char* Controller::getSubjectBuffer() { return
custom_quert_buffers["Subject"].data(); }
char* Controller::getFirstNameBuffer() { return
custom_quert_buffers["FirstName"].data(); }
char* Controller::getLastNameBuffer() { return
custom_quert_buffers["LastName"].data(); }
char* Controller::getGroupBuffer() { return
custom_quert_buffers["Group"].data(); }

```

```

char* Controller::getAttendanceDateFromBuffer() { return
custom_quert_buffers["AttendanceDateFrom"].data(); }
char* Controller::getAttendanceDateToBuffer() { return
custom_quert_buffers["AttendanceDateTo"].data(); }
char* Controller::getFacultyBuffer() { return
custom_quert_buffers["Faculty"].data(); }

int* Controller::getRandomDataCountPtr() { return &randomDataCount; }
int* Controller::getDeleteRowsFromPtr() { return &deleteRowsFrom; }
int* Controller::getDeleteRowsToPtr() { return &deleteRowsTo; }
bool* Controller::getIsSegmentToDeletePtr() { return &isSegmentToDelete; }

```

## Model.h

```

#pragma once

#include <pqxx/pqxx>

#include <algorithm>
#include <string>
#include <iostream>
#include <map>
#include <stdexcept>

namespace mvc {

    class Model {
    private:
        std::unique_ptr<pqxx::connection> conn;
        std::vector<std::string> tableNames;
        std::map<std::string, std::vector<std::string>> columnNames;
        std::pair<std::string, pqxx::result> tableData;

        Model();

        bool isForeignKey(const std::string& tableName, const std::string&
columnName);

        const pqxx::result requestTableData(const std::string& TableName);

        std::map<std::string, std::string> getPrimaryKeyInfo(pqxx::work&
worker, const std::string& tableName);

    public:

        static Model& getInstance() {
            static Model instance;
            return instance;
        }

        class Spawner {
            static Model& instance() {
                return getInstance();
            }
        }

        friend class Controller;
    };

    Model(const Model&) = delete;
    void operator=(const Model&) = delete;

    bool isConnectionReady();

    void reconnect();

```

```

        const std::vector<std::string>& getTableColumnsNames(const
std::string& TableName);

        const pqxx::result& getTableData(const std::string& TableName);

        const std::vector<std::string>& getTableNames();

        void generateRandomDataForTable(const std::string& tableName, int
rowCount);

        void deleteDataRange(const std::string& tableName, const
pqxx::result& dataToDelete, int rowFrom, int rowTo);

        void updateDataCell(const std::string& tableName, pqxx::row dataRow,
unsigned int changedCellNumber, const std::string& dataToChange);

        void clearAllTables(const std::string currentTableName);

        void insertData(const std::string& tableName, const
std::map<std::string, std::string>& data);

        pqxx::result executeCustomQuery(const std::string& request,
pqxx::params params);

    };
}

```

## model.cpp

```

#include "Model.h"

using namespace mvc;

Model::Model() {
    try {
        std::string cS = "host=localhost port=8040 dbname=Lab1 user=postgres
password=4242";
        conn = std::make_unique<pqxx::connection>(cS);

        pqxx::work worker(*conn);
        pqxx::result res = worker.exec("SELECT tablename FROM pg_tables WHERE
schemaname = 'public' ORDER BY tablename;");

        tableNames.clear();
        for (auto row : res) {
            tableNames.push_back(row[0].as<std::string>());
        }
    }
    catch (const std::exception& e) {
        throw std::runtime_error(e.what());
    }
}

bool Model::isForeignKey(const std::string& tableName, const std::string&
columnName)
{
    try {
        pqxx::nontransaction worker(*conn);

        const std::string sql =
            "SELECT 1 FROM information_schema.table_constraints AS tc "

```

```

        "JOIN information_schema.key_column_usage AS kcu ON
tc.constraint_name = kcu.constraint_name AND tc.table_schema =
kcu.table_schema "
        "WHERE tc.constraint_type = 'FOREIGN KEY' AND tc.table_schema = "
+ worker.quote_name("public") +
        " AND tc.table_name = " + worker.quote_name(tableName) + " AND
kcu.column_name = " + worker.quote_name(columnName) + " LIMIT 1;";

        pqxx::result res = worker.exec(sql);

        return !res.empty();
    }
    catch (const std::exception& e) {
        throw std::runtime_error(e.what());
        return false;
    }
}

const pqxx::result Model::requestTableData(const std::string& tableName) {
    if (!conn)
        reconnect();
    try {
        pqxx::work worker(*conn);
        pqxx::result res = worker.exec("SELECT * FROM " +
worker.quote_name(tableName));
        tableData = { tableName, res };
        return tableData.second;
    }
    catch (const std::exception& e) {
        throw std::runtime_error(e.what());
    }
}

std::map<std::string, std::string> Model::getPrimaryKeyInfo(pqxx::work&
worker, const std::string& tableName) {
    std::string pk_query =
        "SELECT kcu.column_name, c.data_type "
        "FROM information_schema.table_constraints AS tc "
        "JOIN information_schema.key_column_usage AS kcu "
        " ON tc.constraint_name = kcu.constraint_name AND tc.table_schema =
kcu.table_schema "
        "JOIN information_schema.columns AS c "
        " ON kcu.table_name = c.table_name AND kcu.table_schema =
c.table_schema AND kcu.column_name = c.column_name "
        "WHERE tc.constraint_type = 'PRIMARY KEY' "
        " AND tc.table_name = " + worker.quote(tableName) +
        " ORDER BY kcu.ordinal_position;";

    pqxx::result pk_res = worker.exec(pk_query);

    std::map<std::string, std::string> pkInfo;
    for (const auto& row : pk_res) {
        pkInfo[row[0].as<std::string>()] = row[1].as<std::string>();
    }
    return pkInfo;
}

bool Model::isConnectionReady() {
    if (!conn)
        return false;
    return true;
}

void Model::reconnect() {

```

```

    try {
        std::string cS = "host=localhost port=8040 dbname=Lab1 user=postgres
password=4242";
        conn = std::make_unique<pqxx::connection>(cS);
    }
    catch (const std::exception& e) {
        throw std::runtime_error(e.what());
    }
}

const std::vector<std::string>& Model::getTableNames() {
    return tableName;
}

const std::vector<std::string>& Model::getTableColumnsNames(const
std::string& tableName) {

    if (!conn)
        reconnect();

    auto it = columnNames.find(tableName);
    if (it != columnNames.end())
        return it->second;

    std::vector<std::string> names;
    try {
        pqxx::work worker(*conn);
        pqxx::result res = worker.exec("SELECT column_name FROM
information_schema.columns WHERE table_name = " + worker.quote(tableName) +
        " AND table_schema = 'public' ORDER BY ordinal_position;");

        for (const auto& row : res)
            names.push_back(row[0].as<std::string>());

    }
    catch (const std::exception& e) {
        throw std::runtime_error(e.what());
        return names;
    }

    columnNames[tableName] = names;

    return columnNames.at(tableName);
}

const pqxx::result& Model::getTableData(const std::string& tableName) {
    if (tableData.first != tableName)
        return requestTableData(tableName);
    return tableData.second;
}

void Model::generateRandomDataForTable(const std::string& tableName, int
rowCount) {
    try {
        if (!conn)
            reconnect();

        pqxx::work txn(*conn);

        if (tableName.find("teacher_subject") != std::string::npos)
        {
            // Перевіряємо, чи є вхідні дані
            pqxx::result t_count = txn.exec("SELECT COUNT(*) FROM Teacher;");
            pqxx::result s_count = txn.exec("SELECT COUNT(*) FROM Subject;");

```

```

        if (t_count[0][0].as<int>() == 0 || s_count[0][0].as<int>() == 0)
        {
            throw std::runtime_error("cannot generate links - Teacher or
Subject table is empty.");
        }

        std::string insertSQL =
            "INSERT INTO teacher_subject (teacher_id, subject_id) "
            "SELECT t.teacher_id, s.subject_id "
            "FROM Teacher t "
            "CROSS JOIN Subject s "
            "LEFT JOIN teacher_subject ts "
            "  ON t.teacher_id = ts.teacher_id AND s.subject_id =
ts.subject_id "
            "WHERE ts.teacher_id IS NULL "
            "ORDER BY random() "
            "LIMIT $1";

        txn.exec(insertSQL, rowsCount);
        txn.commit();
        requestTableData(tableName);
        return;
    }

    pqxx::result fks = txn.exec("(SELECT kcu.column_name, ccu.table_name
AS foreign_table, ccu.column_name AS foreign_column "
        "FROM information_schema.table_constraints AS tc JOIN
information_schema.key_column_usage AS kcu ON tc.constraint_name =
kcu.constraint_name "
        "JOIN information_schema.constraint_column_usage AS ccu ON
ccu.constraint_name = tc.constraint_name "
        "WHERE tc.table_name = " + txn.quote(tableName) + " AND
tc.constraint_type = 'FOREIGN KEY' AND tc.table_schema = 'public')");

    std::vector<std::string> missingTables;
    for (auto const& row : fks) {
        std::string refTable = row["foreign_table"].c_str();
        pqxx::result count = txn.exec("SELECT COUNT(*) FROM " +
txn.quote_name(refTable) + ";");
        if (count[0][0].as<int>() == 0)
            missingTables.push_back(refTable);
    }

    if (!missingTables.empty()) {
        std::string msg = "missing data in related tables: ";
        for (size_t i = 0; i < missingTables.size(); ++i) {
            msg += missingTables[i] + (i + 1 < missingTables.size() ? ",
" : "");
        }
        throw std::runtime_error(msg);
    }

    pqxx::result all_cols = txn.exec("(SELECT column_name, data_type,
character_maximum_length, column_default FROM information_schema.columns "
        "WHERE table_name = " + txn.quote(tableName) + " AND table_schema
= 'public' ORDER BY ordinal position)");

    std::string colsList, valsList;
    std::vector<pqxx::row> colsToInsert;

    for (auto const& row : all_cols) {
        if (row["column_default"].is_null()) {
            colsToInsert.push_back(row);
        }
    }

```

```

    }

    if (colsToInsert.empty()) {
        throw std::runtime_error("No columns to insert");
    }

    for (size_t i = 0; i < colsToInsert.size(); ++i) {
        colsList +=
txn.quote_name(colsToInsert[i]["column_name"].c_str());
        if (i + 1 < colsToInsert.size()) colsList += ", ";
    }

    for (size_t i = 0; i < colsToInsert.size(); ++i) {
        std::string colName = colsToInsert[i]["column_name"].c_str();
        std::string colType = colsToInsert[i]["data_type"].c_str();
        auto collength = colsToInsert[i]["character_maximum_length"];
        bool isFK = false;
        std::string fkSelect;

        for (auto const& fk : fks) {
            if (fk["column_name"].c_str() == colName) {
                isFK = true;
                fkSelect = "(SELECT " +
txn.quote_name(fk["foreign_column"].c_str()) +
                    " FROM " +
txn.quote_name(fk["foreign_table"].c_str()) +
                    " ORDER BY random() LIMIT 1 OFFSET (s.i * 0))";

                break;
            }
        }

        if (isFK) {
            valsList += fkSelect;
        }
        else if (colType.find("char") != std::string::npos || colType ==
"text") {

            std::string generator_sql;
            bool apply_substr = true;

            if (colName.find("first_name") != std::string::npos) {
                generator_sql = reinterpret_cast<const
char*>(u8"(ARRAY['Іван', 'Петро', 'Марія', 'Ярослав', 'Анна', 'Володимир', "
                    u8"Василь', 'Ганна', 'Вікторія', 'Анастасія',
                    'Денис', 'Артем', 'Руслан', 'Михайло', 'Микита'])[floor(random() * 15 +
                    1)]");
            }
            else if (colName.find("last_name") != std::string::npos) {
                generator_sql = reinterpret_cast<const
char*>(u8"(ARRAY['Шевченко', 'Коваленко', 'Петренко', 'Іваненко', 'Бойко',
                    'Мельник', "
                    u8"Одарченко', 'Порошенко', 'Мироненко', 'Сектим',
                    'Гончаренко', 'Винник', 'Боженко', 'Файнберг'])[floor(random() * 14 + 1)]");
            }
            else if (colName.find("status") != std::string::npos) {
                generator_sql = reinterpret_cast<const
char*>(u8"(ARRAY['Присутній', 'Відсутній'])[floor(random() * 2 + 1)]");
            }
            else if (colName.find("class_type") != std::string::npos) {
                generator_sql = reinterpret_cast<const
char*>(u8"(ARRAY['Практика', 'Лекція', 'Лабораторна'])[floor(random() * 3 +
                    1)]");
            }
        }
    }
}

```

```

    }
    else if (colName.find("email") != std::string::npos) {
        generator_sql = "substr(md5(random()::text), 1, 5) || '.'
|| substr(md5(random()::text), 1, 7) || "
        "(ARRAY['@gmail.com', '@ukr.net',
'@kpi.ua'])[floor(random() * 3 + 1)]";
    }
    else if (colName.find("group_name") != std::string::npos) {
        generator_sql = reinterpret_cast<const
char*>(u8"(ARRAY['KB', 'KM', 'KII'])[floor(random() * 3 + 1)] || '-' || "
        "floor(random() * 90000 + 10000)::int || '-' ||
gen_random_uuid()::text");
    }
    else if (colName.find("subject_name") != std::string::npos) {
        generator_sql = reinterpret_cast<const
char*>(u8"(ARRAY['COHC', 'AMO', 'KE', 'ЗІКC', 'БДЗУ', 'KC'])[floor(random() *
6 + 1)] "
        "|| '-' || gen_random_uuid()::text");
    }
    else if (colName.find("faculty") != std::string::npos) {
        generator_sql = reinterpret_cast<const
char*>(u8"(ARRAY['ФІМ', 'ФМФ', 'ФІОТ'])[floor(random() * 3 + 1)]");
    }

    else {
        std::string len = "8";
        if (!colLength.is_null()) {
            int dbLen = colLength.as<int>();
            len = (std::to_string)((std::min)(dbLen, 32));
        }
        generator_sql = "substr(md5(random()::text), 1, " + len +
")";
        apply_substr = false;
    }

    if (apply_substr && !colLength.is_null()) {
        valsList += "substr(" + generator_sql + ", 1, " +
colLength.as<std::string>() + ")";
    }
    else {
        valsList += generator_sql;
    }
}
else if (colType.find("int") != std::string::npos) {
    valsList += "floor(random()*1000)::int";
}
else if (colType.find("date") != std::string::npos ||
colType.find("timestamp") != std::string::npos) {
    valsList += "now() - (random() * interval '10 years')";
}
else if (colType == "boolean") {
    valsList += "random() > 0.5";
}
else {
    valsList += "NULL";
}

if (i + 1 < colsToInsert.size()) valsList += ", ";
}

std::string insertSQL = "INSERT INTO " + txn.quote_name(tableName) +
" (" + colsList + ") " +

```

```

        " SELECT " + valsList +
        " FROM generate_series(1, " + std::to_string(rowsCount) + ") AS
s(i)";

    txn.exec(insertSQL.c_str()).no_rows();

    txn.commit();
    requestTableData(tableName);
}
catch (std::exception const& e) {
    throw std::runtime_error(e.what());
}
}

void Model::deleteDataRange(const std::string& tableName, const pqxx::result&
dataToDelete, int rowFrom, int rowTo)
{
    if (rowFrom > rowTo) {
        std::swap(rowFrom, rowTo);
    }
    if (dataToDelete.empty()) {
        throw std::runtime_error("No data to delete");
    }

    std::string tempTableName = "temp_pks_to_delete_" +
std::to_string(std::time(nullptr));

    try {
        pqxx::work worker(*conn);

        std::map<std::string, std::string> pkInfo = getPrimaryKeyInfo(worker,
tableName);
        if (pkInfo.empty()) {
            throw std::runtime_error("No PK in table");
        }

        std::vector<int> pkIndicesInResult;
        std::string pkColsForCreate;

        std::vector<std::string> pkColNamesForCopy;

        std::string pkColsForJoin;

        try {
            for (const auto& pair : pkInfo) {
                const std::string& colName = pair.first;
                const std::string& colType = pair.second;

                pkIndicesInResult.push_back(dataToDelete.column_number(colNam
e));

                std::string quotedName = worker.quote_name(colName);

                pkColsForCreate += quotedName + " " + colType + ",";
                pkColNamesForCopy.push_back(colName);
                pkColsForJoin += "t." + quotedName + " = tmp." + quotedName +
" AND ";
            }
        }
        catch (const std::exception& e) {
            throw std::runtime_error("Can not find PK column: " +
std::string(e.what()));
        }

        pkColsForCreate.pop_back();

```

```

        pkColsForJoin.resize(pkColsForJoin.size() - 5);

        std::string createTempTableSql = "CREATE TEMPORARY TABLE " +
worker.quote_name(tempTableName) +
        " (" + pkColsForCreate + ") ON COMMIT DROP;";
        worker.exec(createTempTableSql);

        std::string full = tempTableName + "(";
        for (size_t i = 0; i < pkColNamesForCopy.size(); ++i)
        {
            full += worker.quote_name(pkColNamesForCopy[i]);
            if (i + 1 != pkColNamesForCopy.size()) full += ",";
        }
        full += ")";

        auto copier = pqxx::stream_to::raw_table(worker, full);

        for (int i = rowFrom; i <= rowTo; ++i) {
            const auto& row = dataToDelete[i];
            std::vector<std::string> values;
            for (size_t j = 0; j < pkIndicesInResult.size(); ++j) {
                const auto& field = row[pkIndicesInResult[j]];
                if (field.is_null()) {
                    values.push_back("\\N");
                }
                else {
                    values.push_back(field.c_str());
                }
            }
            copier.write_values(values);
        }
        copier.complete();

        std::string deleteSql = "DELETE FROM " + worker.quote_name(tableName)
+ " t "
            "USING " + worker.quote_name(tempTableName) + " tmp "
            "WHERE " + pkColsForJoin;
        worker.exec(deleteSql);

        worker.commit();
        requestTableData(tableName);
    }
    catch (const std::exception& e) {
        throw std::runtime_error("Deleting data error: " +
std::string(e.what()));
    }
}

void Model::updateDataCell(const std::string& tableName, pqxx::row dataRow,
unsigned int changedCellNumber, const std::string& dataToChange) {
    if (dataRow.size() <= 0) {
        throw std::runtime_error("Nothing to update?");
        return;
    }
    try {
        std::vector<std::string> colNames = getTableColumnsNames(tableName);

        if (colNames[changedCellNumber].find("status") != std::string::npos
&& tableName.find("attendance") != std::string::npos) {
            if (dataToChange != "Присутній" && dataToChange != "Відсутній") {
                throw std::runtime_error("Wrong status! Should be 'Присутній'
or 'Відсутній'");
            }
        }
    }
}

```

```

        pqxx::work worker(*conn);
        std::string request = "UPDATE " + worker.quote_name(tableName) + "
SET " + colNames[changedCellNumber] + " = " + worker.quote(dataToChange) + "
WHERE ";
        unsigned int i = 0;
        for (const auto& element : dataRow) {
            if (i != changedCellNumber) {
                request += worker.quote_name(colNames[i]);

                if (element.is_null()) {
                    request += " IS NULL AND ";
                }
                else {
                    request += " = " +
worker.quote(element.as<std::string>()) + " AND ";
                }
            }
            i++;
        }
        if (request.size() >= 5) request.resize(request.size() - 5);

        worker.exec(request);
        worker.commit();
        requestTableData(tableName);
    }
    catch (const std::exception& e) {
        throw std::runtime_error(e.what());
    }
}

void Model::clearAllTables(const std::string currentTableName) {
    try {
        pqxx::work worker(*conn);
        std::string request = "TRUNCATE TABLE ";

        std::vector<std::string> names = getTableNames();
        for (const auto& name : names)
            request += name + ", ";
        request.pop_back();
        request.pop_back();
        request += " RESTART IDENTITY CASCADE;";

        worker.exec(request);
        worker.commit();
        requestTableData(currentTableName);
    }
    catch (const std::exception& e) {
        throw std::runtime_error(e.what());
    }
}

void Model::insertData(const std::string& tableName, const
std::map<std::string, std::string>& data) {
    if (data.empty())
        throw std::runtime_error("No data to insert");

    try {
        pqxx::work worker(*conn);
        std::string request = "INSERT INTO " + worker.quote_name(tableName) +
" (";

        std::string cols = "", vals = "";
        for (const auto& pair : data) {

```

```

        if (pair.first.find("status") != std::string::npos &&
            tableName.find("attendance") != std::string::npos) {
            if (pair.second != "Присутній" && pair.second != "Відсутній")
            {
                throw std::runtime_error("Wrong status! Should be
'Присутній' or 'Відсутній'");
            }
        }
        cols += worker.quote_name(pair.first) + ",";
        vals += worker.quote(pair.second) + ",";
    }
    cols.pop_back();
    vals.pop_back();
    request += cols + ") VALUES (" + vals + ")";

    worker.exec(request);
    worker.commit();
    requestTableData(tableName);
}
catch (const std::exception& e) {
    throw std::runtime_error(e.what());
}
}

pqxx::result Model::executeCustomQuery(const std::string& request,
pqxx::params params) {
    try {
        pqxx::work worker(*conn);
        return worker.exec(request, params);
    }
    catch (const std::exception& e) {
        throw std::runtime_error(e.what());
    }
}
}

```

## View.h

```

#pragma once

#include <iostream>

#include "imgui/imgui.h"
#include "imgui-sfml/imgui-SFML.h"
#include "SFML/Graphics.hpp"
#include "pqxx/pqxx"

#include "r_handler.h"

namespace mvc {

    static unsigned int WINDOW_SIZE_X = 1280, WINDOW_SIZE_Y = 720;
    class Controller;

    class View {
    private:
        sf::RenderWindow window;
        ResponseHandler& response;
        mvc::Controller* controller;

        std::vector<std::string> columnsNames = {};
        std::vector<std::string> tableNames = {};

        View(ResponseHandler& r);

        void Draw();
    };
}

```

```

        void HandleInput();
        void Update(sf::Clock& delta);

        void table(const std::string& name);
        void insert();
        void misc(const std::string& tableName);
        void tabs();
        void mainWindow();
        void searchWindow();

        static View& getInstance(ResponseHandler& r) {
            static View instance(r);
            return instance;
        }

public:
    ~View();

    View(const View&) = delete;
    void operator=(const View&) = delete;

    class Spawner {
        static View& instance(ResponseHandler& r) {
            return getInstance(r);
        }

        friend class Controller;
    };

    void setController(mvc::Controller* Controller) {
        controller = Controller;
    }

    void Start();
};
}

```

## view.cpp

```

#include "View.h"
#include "Controller.h"

using namespace mvc;

View::View(ResponseHandler& r) : window(sf::VideoMode({ WINDOW_SIZEX,
WINDOW_SIZEY })), "PostgreSQL manager"), response(r) {
    window.setFramerateLimit(120);
    ImGui::SFML::Init(window);

    ImGuiIO& io = ImGui::GetIO();
    io.Fonts->Clear();
    io.Fonts->AddFontFromFileTTF("Arial.ttf", 16.0f, nullptr, io.Fonts->GetGlyphRangesCyrillic());

    if (!ImGui::SFML::UpdateFontTexture()) {
        std::cerr << "View: font update error!" << std::endl;
    }
}

View::~~View() {
    if (ImGui::GetCurrentContext() != nullptr)
    {
        ImGui::SFML::Shutdown();
    }
}

```

```

    }
}

void View::Draw() {
    window.clear();

    ImGui::SFML::Render(window);

    window.display();
}

void View::HandleInput() {
    while (const std::optional event = window.pollEvent()) {
        ImGui::SFML::ProcessEvent(window, *event);

        if (event->is<sf::Event::Closed>()) {
            window.close();
        }

        if (event->is<sf::Event::Resized>()) {
            const auto& newSize = event->getIf<sf::Event::Resized>()->size;
            WINDOW_SIZEEX = newSize.x;
            WINDOW_SIZEY = newSize.y;
        }
    }
}

void View::table(const std::string& name) {
    controller->current_table_name = name;
    columnsNames = controller->getTableColumnsNames(name);

    const pqxx::result& res = controller->getTableData(name);

    if (ImGui::BeginTable("activeTable", columnsNames.size() + 1,
ImGuiTableFlags_Borders |
    ImGuiTableFlags_ScrollY | ImGuiTableFlags_RowBg, ImVec2(0,
WINDOW_SIZEY * 0.78))) {

        ImGui::TableSetupScrollFreeze(0, 1);
        ImGui::TableSetupColumn("Nu", ImGuiTableColumnFlags_WidthFixed);
        for (const auto& columnName : columnsNames)
            ImGui::TableSetupColumn(columnName.c_str());
        ImGui::TableHeadersRow();

        char label_id[256];

        ImGuiListClipper clipper;
        clipper.Begin(res.size());

        while (clipper.Step()) {
            for (int current_row_index = clipper.DisplayStart;
current_row_index < clipper.DisplayEnd; ++current_row_index) {
                const auto& row = res[current_row_index];
                ImGui::TableNextRow();

                ImGui::TableNextColumn();
                sprintf_s(label_id, "%d##nu_%d", current_row_index + 1,
current_row_index);
                bool is_row_selected = controller-
>isCellEditing(current_row_index, -1);
                if (ImGui::Selectable(label_id, is_row_selected,
ImGuiSelectableFlags_AllowDoubleClick)) {
                    controller->onCellSelected(current_row_index, -1);
                }
            }
        }
    }
}

```

```

        for (int i = 0; i < columnsNames.size(); i++) {
            ImGui::TableNextColumn();

            std::string cell_content = row[i].is_null() ? "" :
row[i].as<std::string>();
            sprintf_s(label_id, "%s##%d_%d", cell_content.c_str(),
current_row_index, i);

            if (controller->isCellEditing(current_row_index, i)) {
                ImGui::SetKeyboardFocusHere();

                if (ImGui::InputText("Enter/ESC", controller-
>getEditBuffer(), controller->getBufferSize(),
ImGuiInputTextFlags_EnterReturnsTrue |
                ImGuiInputTextFlags_AutoSelectAll |
ImGuiInputTextFlags_EscapeClearsAll))
                {
                    controller->onCellEditConfirmed();
                }
                else if (ImGui::IsItemFocused() &&
ImGui::IsKeyPressed(ImGuiKey_Escape))
                {
                    controller->onCellEditCanceled();
                }
            }
            else {
                if (ImGui::Selectable(label_id, controller-
>isCellSelected(current_row_index, i),
ImGuiSelectableFlags_AllowDoubleClick))
                {
                    controller->onCellSelected(current_row_index, i);

                    if (ImGui::IsMouseDoubleClicked(0)) {
                        controller-
>onCellDoubleClicked(current_row_index, i, cell_content);
                    }
                }
            }
        }
    }
    ImGui::EndTable();
}

void View::insert() {
    if (ImGui::BeginTable("additionalTable", columnsNames.size()+1,
ImGuiTableFlags_RowBg)) {
        ImGui::TableSetupColumn("Actions", ImGuiTableColumnFlags_WidthFixed);
        ImGui::TableNextRow();
        ImGui::TableNextColumn();
        if (ImGui::SmallButton("Clear")) { controller->clearInsertBuffers(); }

        if (ImGui::SmallButton("Instert")) { controller->onInsertClicked(); }
        for (const auto& colName : columnsNames) {
            ImGui::TableNextColumn();
            ImGui::PushItemWidth(-1);
            ImGui::InputText(std::string("Enter/ECS##insert_" +
colName).c_str(), controller->getInsertBuffer(colName), controller-
>getBufferSize(),
                ImGuiInputTextFlags_EscapeClearsAll);
            ImGui::PopItemWidth();
        }
    }
}

```

```

    }

    ImGui::EndTable();
}

void View::misc(const std::string& tableName) {

    insert();

    if (ImGui::Button("Add Random Data")) {
        controller->onAddRandomDataClicked();
    }

    ImGui::SetNextItemWidth(WINDOW_SIZE_X / 7);
    ImGui::SameLine();
    ImGui::InputInt("Amount of Random Data", controller-
>getRandomDataCountPtr());

    ImGui::SameLine(0.0f, 30.0f);
    ImGui::TextColored(response.isLastCommitWasAnError() ? ImVec4(0.9, 0.1,
0.1, 1) : ImVec4(0.1, 0.9, 0.1, 1), "%s", response.getResponse().c_str());

    bool* isSegment = controller->getIsSegmentToDeletePtr();

    if (ImGui::Button((*isSegment ? "Delete rows" : "Delete row"))) {
        controller->onDeleteRowsClicked();
    }

    ImGui::SameLine();
    ImGui::Checkbox("segment", isSegment);

    ImGui::SameLine();
    ImGui::SetNextItemWidth(WINDOW_SIZE_X / 8);
    if (*isSegment) {
        ImGui::InputInt("From", controller->getDeleteRowsFromPtr());
        ImGui::SameLine();
        ImGui::SetNextItemWidth(WINDOW_SIZE_X / 8);
        ImGui::InputInt("To", controller->getDeleteRowsToPtr());
    }
    else {
        ImGui::InputInt("Number (Nu)", controller->getDeleteRowsFromPtr());
    }

    ImGui::SameLine();
    ImGui::SetCursorPosX( WINDOW_SIZE_X - ImGui::CalcTextSize("Clear all
tables").x - 15);
    if (ImGui::Button("Clear all tables"))
        controller->onClearAllClicked();

    ImGui::SetCursorPosX({ WINDOW_SIZE_X - (ImGui::CalcTextSize("Search").x +
13), 5 });
    if (ImGui::SmallButton("Search"))
        controller->onOpenSearchWindowClicked();
}

void View::tabs() {
    if (ImGui::BeginTabBar("TablesBar")) {

        tableNames = controller->getTableNames();

        for (const std::string& name : tableNames) {
            if (ImGui::BeginTabItem(name.c_str())) {

```

```

        table(name);
        misc(name);

        ImGui::EndTabItem();
    }
}

ImGui::EndTabBar();
}

}

void View::mainWindow() {
    ImGui::SetNextWindowPos(ImVec2(0, 0), ImGuiCond_Always);
    ImGui::SetNextWindowSize(ImVec2(window.getSize().x, window.getSize().y),
ImGuiCond_Always);

    if (ImGui::Begin("Main begin", nullptr, ImGuiWindowFlags_NoTitleBar |
ImGuiWindowFlags_NoCollapse | ImGuiWindowFlags_NoResize |
    ImGuiWindowFlags_NoMove | ImGuiWindowFlags_NoBringToFrontOnFocus |
ImGuiWindowFlags_NoNavFocus)) {

        tabs();

        ImGui::End();
    }
}

void View::searchWindow() {
    if (!controller->isSearchWindowOpen())
        return;

    ImGui::OpenPopup("Search 6_6");
    ImGui::SetNextWindowPos(ImVec2( WINDOW_SIZEX / 2,WINDOW_SIZEY / 2 ),
ImGuiCond_Appearing, ImVec2(0.5f, 0.5f));

    if (ImGui::BeginPopupModal("Search 6_6", NULL,
ImGuiWindowFlags_AlwaysAutoResize)) {

        ImGui::Text("Advanced search");
        ImGui::Separator();
        ImGui::Text("List of teachers that leads the subject: ");
        ImGui::Text("Enter subject name ");
        ImGui::SameLine();
        ImGui::InputText("##subject_input", controller->getSubjectBuffer(),
controller->getBufferSize());
        ImGui::Text("Enter teachers first/last name: ");
        ImGui::SameLine();
        ImGui::InputText("##first_name_input", controller-
>getFirstNameBuffer(), controller->getBufferSize());
        ImGui::SameLine();
        ImGui::InputText("##last_name_input", controller-
>getLastNameBuffer(), controller->getBufferSize());
        ImGui::TextDisabled("Empty teachers fields means that it will search
for teacher with any first/last name");
        if (ImGui::Button("Search##1"))
            controller->on1SearchRequestClicked();

        ImGui::Separator();
        ImGui::Text("Student attendance in groups by dates(YYYY-MM-DD): ");
        ImGui::Text("Enter group name ");
        ImGui::SameLine();
        ImGui::InputText("##group_input", controller->getGroupBuffer(),
controller->getBufferSize());
        ImGui::Text("Enter dates from ");

```

```

        ImGui::SameLine();
        ImGui::InputText("##attendance_date_input_from", controller-
>getAttendanceDateFromBuffer(), controller->getBufferSize());
        ImGui::SameLine();
        ImGui::Text(" and to ");
        ImGui::SameLine();
        ImGui::InputText("##attendance_date_input_to", controller-
>getAttendanceDateToBuffer(), controller->getBufferSize());
        ImGui::TextDisabled("Empty date fields means that it will search
throughout the entire time");
        if (ImGui::Button("Search##2"))
            controller->on2SearchRequestClicked();

        ImGui::Separator();
        ImGui::Text("Number of students who missed classes by faculty: ");
        ImGui::Text("Enter faculty: ");
        ImGui::SameLine();
        ImGui::InputText("##faculty_name", controller->getFacultyBuffer(),
controller->getBufferSize());
        if (ImGui::Button("Search##3"))
            controller->on3SearchRequestClicked();

        ImGui::Separator();

        const pqxx::result& res = controller->search_res;
        if (res.size() <= 0)
            goto A;

        if (ImGui::BeginTable("Search result", res.columns(),
ImGuiTableFlags_ScrollY | ImGuiTableFlags_RowBg, { 0, WINDOW_SIZE_X * 0.15f
})) {

            for (int i = 0; i < res.columns(); ++i) {
                ImGui::TableSetupColumn(res.column_name(i));
            }
            ImGui::TableHeadersRow();

            ImGuiListClipper clipper;
            clipper.Begin(res.size());

            while (clipper.Step()) {
                for (int row_idx = clipper.DisplayStart; row_idx <
clipper.DisplayEnd; ++row_idx) {
                    ImGui::TableNextRow();
                    for (int col_idx = 0; col_idx < res.columns(); ++col_idx)
                    {
                        ImGui::TableNextColumn();

                        // NULL перевірка
                        const auto& field = res[row_idx][col_idx];
                        const char* cell_text = field.is_null() ? "[NULL]" :
field.c_str();

                        ImGui::TextUnformatted(cell_text);
                    }
                }
            }

            ImGui::EndTable();
        }

A:

```

```

        ImGui::TextColored(response.isLastCommitWasAnError() ? ImVec4(0.9f,
0.1f, 0.1f, 1.0f) : ImVec4(0.1f, 0.9f, 0.1f, 1.0f), "%s",
response.getResponse().c_str());

        ImGui::Separator();
        if (ImGui::Button("Close")) {
            controller->onCloseSearchWindowClicked();
            ImGui::CloseCurrentPopup();
        }

        ImGui::EndPopup();
    }
}

void View::Update(sf::Clock& delta) {
    ImGui::SFML::Update(window, delta.restart());

    mainWindow();
    searchWindow();
}

void View::Start() {
    sf::Clock delta;

    while (window.isOpen())
    {
        HandleInput();
        Update(delta);
        Draw();
    }
}

```

## r\_handler.h

```

#pragma once

#include <string>
#include "SFML/System.hpp"
#include <iostream>

class ResponseHandler {
    std::string ResponseMessage;
    sf::Clock timer;
    bool error;

    void commit(const std::string& Message) {
        if (!Message.empty()) {
            ResponseMessage = std::string(error ? ErrorMessage :
SuccessMessage) + ": " + Message;
            timer.restart();
        }
    }

public:
    // in seconds
    static constexpr unsigned int message_time = 5;
    static constexpr const char* SuccessMessage = "Success";
    static constexpr const char* ErrorMessage = "Error";

    ResponseHandler() : error(false), ResponseMessage("Hi user!"), timer() {
        timer.start();
    }

    const std::string getResponse() {
        if (timer.getElapsedTime().asSeconds() < message_time)

```

```
        return ResponseMessage;
    return {};}

}

void commitErrorMessage(const std::string& Message) {
    error = true;
    commit(Message);
}

void commitSuccessMessage(const std::string& Message) {
    error = false;
    commit(Message);
}

bool isLastCommitWasAnError() {
    return error;
}

};
```

## Посилання на гітхаб