



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  
**«Бази даних і засоби управління»**

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

Виконав: студент III курсу

ФПМ групи КВ-23

Зінедін Шайдін

Перевірів:

Метою роботи є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

Завдання роботи полягає у наступному:

1. Реалізувати функції внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

## Виконання роботи

Нормалізована база даних з першої лабораторної роботи

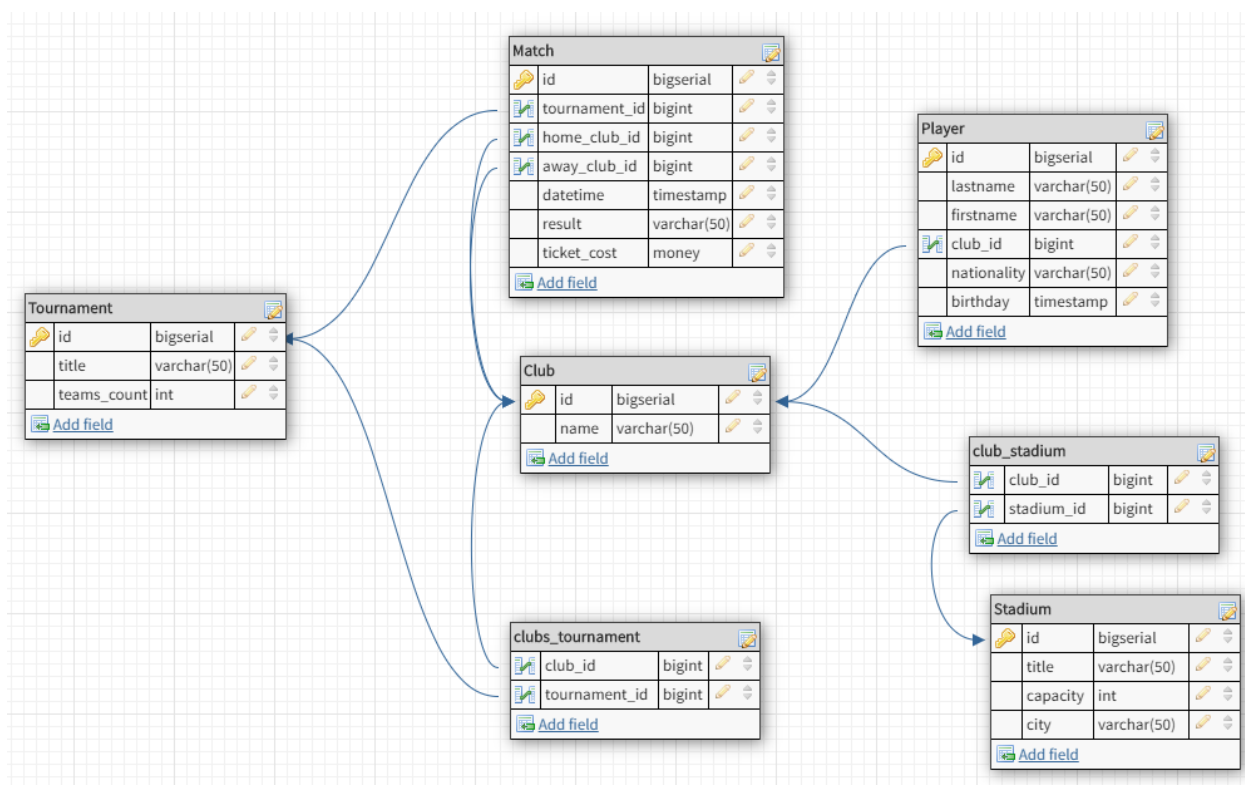


Рисунок 1 - Нормалізована база даних.

## Опис програми

### Структура програми

Програма складається з 4 основних модулів.

1. Файл index.js, який запускає сервер для прийому запитів клієнта;
2. Файли роутингу на серверній частині(для кожної таблиці свій окремий), вони обробляють запити, які надійшли на сервер;
3. Файл Table.js, модуль, для роботи в браузері з таблицею, а саме – пошук, створення, редагування, видалення даних з таблиці бази даних;
4. Файл routes.js для навігації користувача по програмі;
5. Модуль підключення до БД.

#### Навігація по модулях

1. Сервер
  - 1.1. [Запуск сервера.](#)
2. Файли роутингу
  - 2.1. [Для таблиці club;](#)
  - 2.2. [Для таблиці stadium;](#)
  - 2.3. [Для таблиці player;](#)
  - 2.4. [Для таблиці tournament;](#)
  - 2.5. [Для таблиці match;](#)
  - 2.6. [Для таблиці clubs stadiums;](#)
  - 2.7. [Для таблиці clubs tournamets.](#)
3. Модулі для роботи з таблицями.
  - 3.1. [Задання сутностей;](#)
  - 3.2. [Модуль, який відображає їх у браузері.](#)
4. Навігація користувача
  - 4.1. [Модуль навігації;](#)
  - 4.2. [Відображення навігації\(Меню\).](#)
5. [Модуль для підключення до БД.](#)

## Обробка помилок

Для цього завдання кожен SQL запит було виконано в конструкторі try\_catch, приклад:

```
router.get('/club', async(req, res) => {  
  try {  
    const clubs = await pool.query('SELECT * FROM club ORDER BY id;')  
    res.json(clubs.rows)  
  } catch (error) {  
    res.status(400).json('Error');  
    console.log(error.message)  
  }  
})
```

Коли при запиті виникає якась помилка, то сервер поперне користувача відповідь зі статусом помилки(400) і текстом “error”. І коли користувач на свій запит до сервера отримує статус 400 йому на екран буде виведено повідомлення про помилку. Приклад:

```
//makes GET request to server and getting all rows of a table  
const getRows = async (tableName) => {  
  try {  
    const response = await fetch(  
      `${serverUrl}/${tableName}`, {  
        method: 'GET',  
      }  
    )  
  
    const jsonData = await response.json()  
    if(response.status === 400) {  
      window.alert(jsonData)  
      return  
    }  
    setRows(jsonData)  
  } catch (error) {  
    console.log(error.message);  
  }  
}
```

Коли, користувач отримує відповідь, яка має статус 400(тобто помилки), функція window.alert виведе на екран, що сталася помилка.

## Робота delete запитів

В своїй роботі, при видаленні рядків, що мають посилання на зовнішні ключі, я встановлював значення NULL в рядку(якщо він дозволений в таблиці), на який посилався даних зовнішній ключ, якщо NULL не дозволено – видаляв рядок.

Приклад з моєї програми, при видалення рядку з таблиці “club”:

```
router.delete('/club/:id', async(req, res) => {
  try {
    const {id} = req.params
    const qu = `DELETE FROM club_stadium WHERE club_id = ${
id};
    DELETE FROM clubs_tournaments WHERE club_id = ${id};
    DELETE FROM match WHERE home_club_id = ${id} OR away_cl
ub_id = ${id};
    UPDATE player SET club_id = NULL WHERE club_id = ${id};
    DELETE FROM club WHERE id = ${id};`
    const response = await pool.query(qu)

    res.json('Successfully deleted')
  } catch (error) {
    res.status(400).json('Error');
    console.log(error.message)
  }
})
```

Тут можна побачити, що в поле club\_id таблиці player встановлюється значення NULL, а в інших випадках видаляються. Тому, що наприклад, футболіст може існувати без клубу, а футбольний матч, без цього клубу – вже не може.

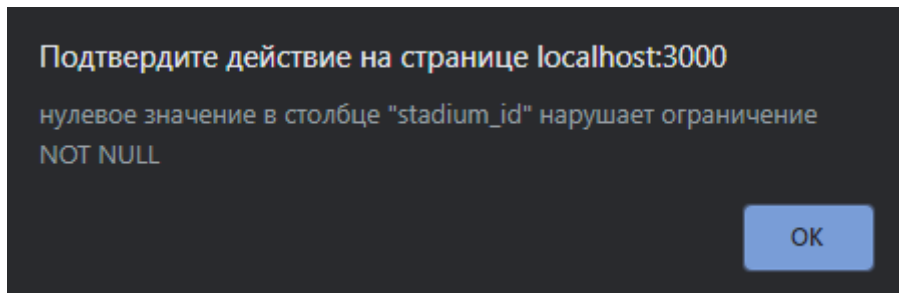
## Дослідження режимів обмеження ON DELETE

### 1. Режим CASCADE

При видаленні запису з таблиці Stadium, запис з таблиці Club\_Stadium видалюється.

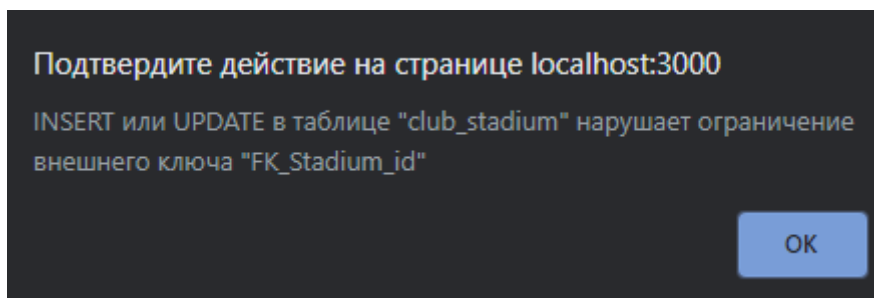
### 2. Режим SET NULL

При видаленні запису з таблиці Stadium, stadium\_id запис з таблиці Club\_Stadium встановлюється в null. Якщо в налаштуваннях таблиці вказати, що stadium\_id не може бути null, то перехоплюємо повідомлення про помилку.



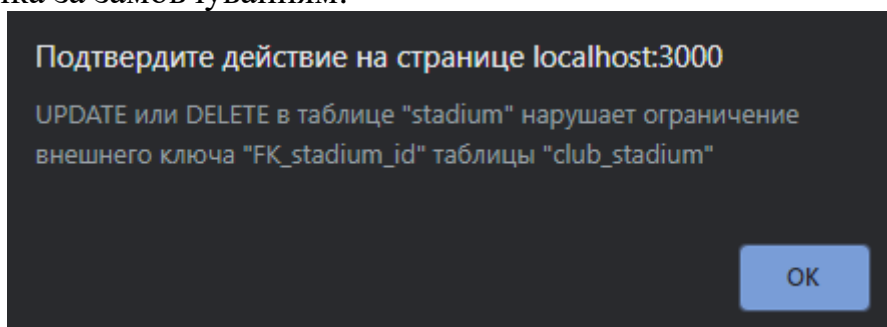
### 3. Режим SET DEFAULT (значення за замовчуванням = 0)

При видаленні запису з таблиці Stadium, перехоплюємо повідомлення про помилку, так як стадіону з id = 0 не існує.



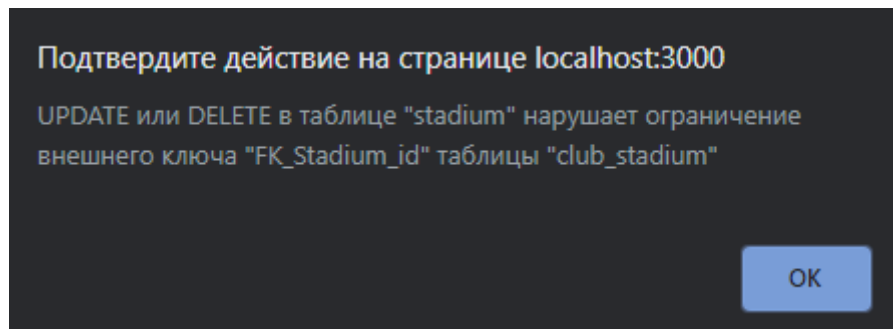
### 4. Режим NO ACTION

При видаленні запису з таблиці Stadium, виникає помилка; це поведінка за замовчуванням.



### 5. Режим *RESTRICT*

При видаленні запису з таблиці Stadium, виникає помилка; це пояснюється тим, що режим **RESTRICT** не дає можливості видалити батьківський рядок, якщо в нього є дочірні.



## Головне меню програми

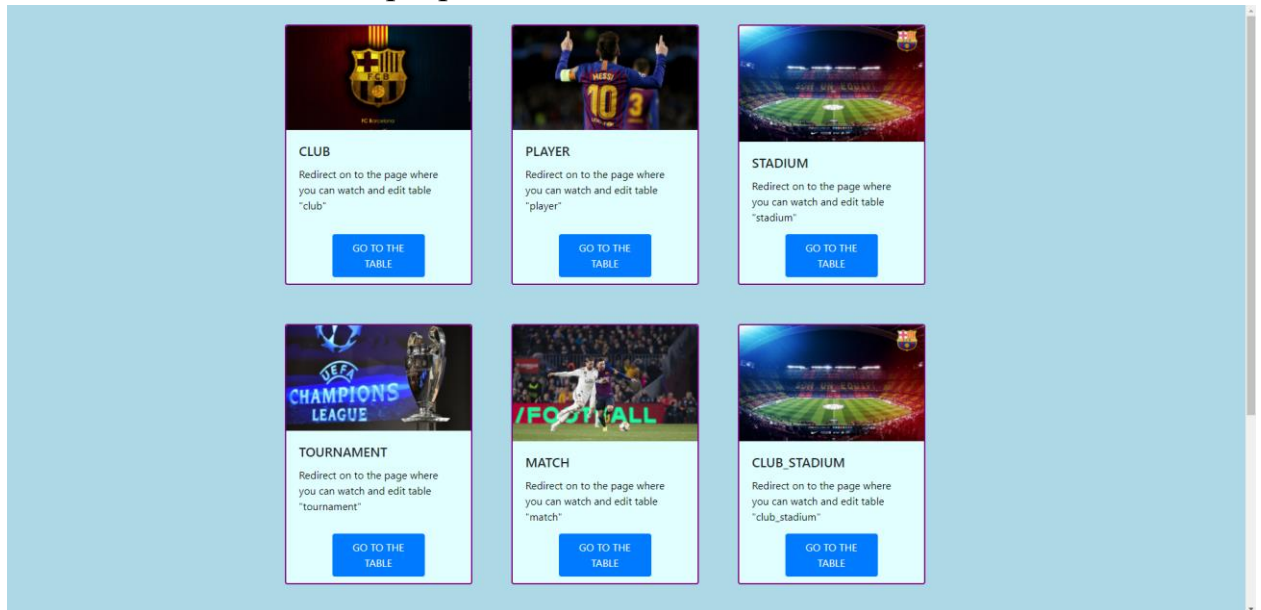


Рисунок 2 - Меню

## Інтерфейс програми

При виконанні графічної роботи для всіх таблиць бази даних було розроблено однаковий інтерфейс. Розглянемо його на прикладі найбільшої з них.

При переході на сторінку таблиці відображаються усі її записи.

id	tournament_id	home_club_id	away_club_id	result	ticket_cost	datetime	Edit	Delete
8	1	1	63	3:0	\$30.00	Oct/25/2024	Edit	Delete
9	1	1	64	3:1	\$30.00	Oct/01/2024	Edit	Delete
10	1	63	65	1:0	\$30.00	Sep/28/2024	Edit	Delete
11	1	64	65		\$30.00	Nov/05/2024	Edit	Delete
12	2	65	66	1:0	\$40.00	Oct/22/2024	Edit	Delete
13	2	66	67	2:4	\$30.00	Oct/29/2024	Edit	Delete
14	2	67	68	4:4	\$40.00	Oct/22/2024	Edit	Delete
15	2	68	69	3:2	\$40.00	Oct/22/2024	Edit	Delete
16	2	69	66	3:1	\$40.00	Oct/25/2024	Edit	Delete
17	7	1	70	2:2	\$45.00	Oct/13/2024	Edit	Delete
18	7	70	71	3:0	\$45.00	Oct/22/2024	Edit	Delete



Рисунок 3 - Приклад Таблиці.

Далі кожен запис можна видалити за допомогою кнопки “Delete”.

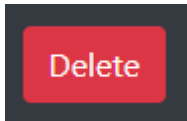
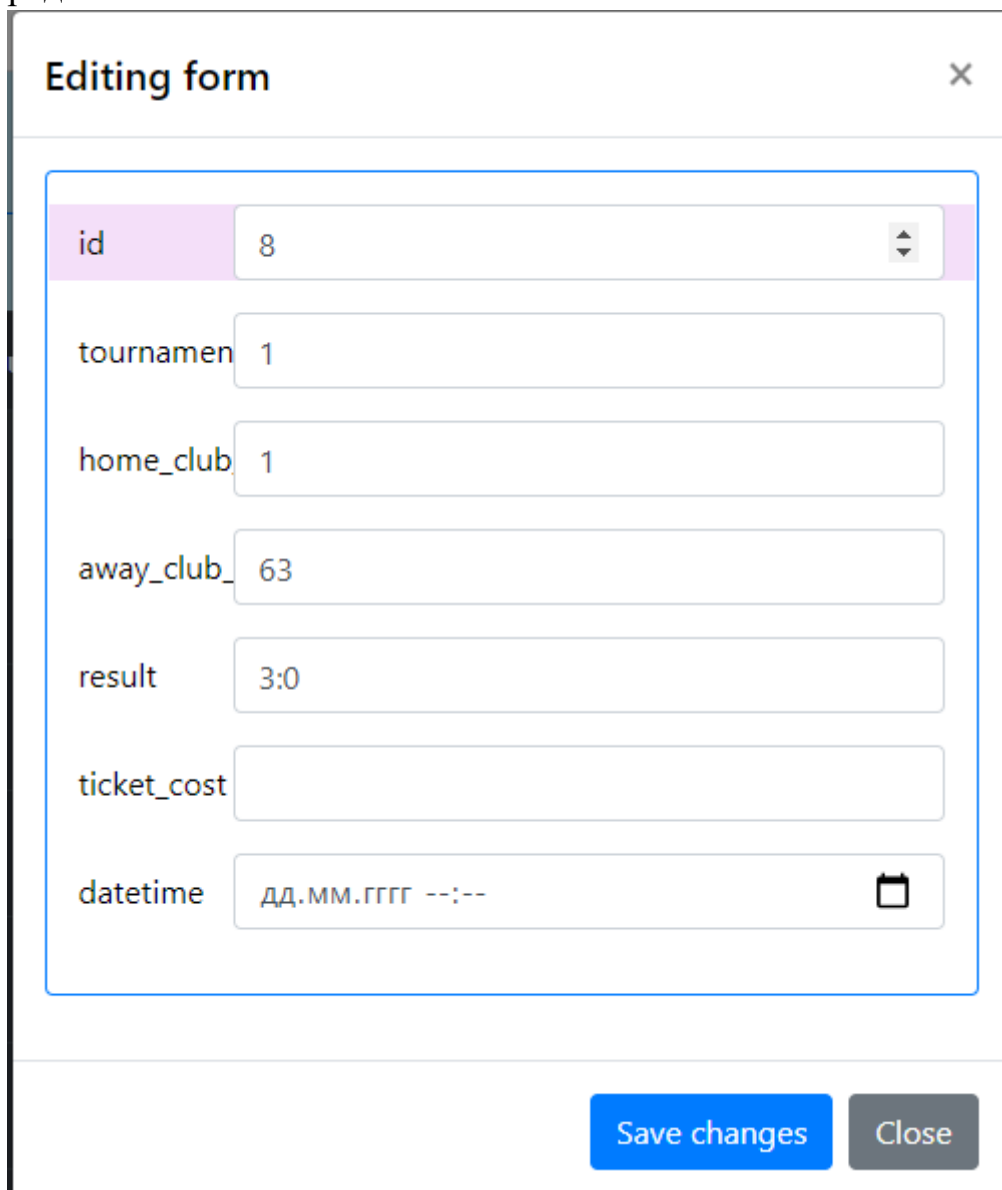


Рисунок 4 - Кнопка видалення.  
Або можна редагувати натиснувши кнопку “Edit”.



Рисунок 5 - Кнопка редагування.  
Після чого відкриється невелике віконце де можна ввести нові дані для цього рядка.

A screenshot of a web application's "Editing form" dialog box. The form is titled "Editing form" with a close button (X) in the top right corner. It contains several input fields: "id" (value 8), "tournamen" (value 1), "home\_club" (value 1), "away\_club\_" (value 63), "result" (value 3:0), "ticket\_cost" (empty), and "datetime" (value DD.MM.YYYY --:-- with a calendar icon). At the bottom right, there are two buttons: "Save changes" (blue) and "Close" (grey).

id	8
tournamen	1
home_club	1
away_club_	63
result	3:0
ticket_cost	
datetime	ДД.ММ.ГГГГ --:--

Рисунок 6 - Форма редагування даних. Також вверху сторінки є форма для пошуку або створення нових записів.

Searching/Creating form

id

tournament\_id

home\_club\_id

away\_club\_id

result

ticket\_cost

datetime

ДД.ММ.РРРР --:--

Find

Create

Рисунок 7 - Пошук рядків. Для того, щоб почати пошук – необхідно ввести інформацію хоча б в одне поле і натиснути кнопку “Find”.

Find

Рисунок 8 - Кнопка пошуку. Після чого, в таблиці будуть значення, які відповідають значенням в цій формі. Для прикладу, знайдемо всі рядки, в яких tournament\_id = 1. Для цього в поле tournament\_id введемо 1 і натиснемо кнопку “Find”. І отримаємо...

id	tournament_id	home_club_id	away_club_id	result	ticket_cost	datetime	Edit	Delete
8	1	1	63	3:0	\$30.00	Oct/25/2024	Edit	Delete
9	1	1	64	3:1	\$30.00	Oct/01/2024	Edit	Delete
10	1	63	65	1:0	\$30.00	Sep/28/2024	Edit	Delete
11	1	64	65		\$30.00	Nov/05/2024	Edit	Delete

Рисунок 9 - Приклад пошуку.

Далі для прикладу створимо, нехай такий рядок.

id	<input type="text"/>
tournament_id	<input type="text" value="7"/>
home_club_id	<input type="text" value="1"/>
away_club_id	<input type="text" value="63"/>
result	<input type="text" value="2:1"/>
ticket_cost	<input type="text" value="40"/>
datetime	<input type="text" value="25.10.2024 20:00"/>
<div><div>Find</div><div>Create</div></div>	

Рисунок 10 - Приклад створення.  
Після чого натиснемо кнопку “Create” і він допишеться в кінець таблиці.

19	7	1	63	2:1	\$40.00	Oct/25/2024	Edit	Delete
----	---	---	----	-----	---------	-------------	------	--------

Рисунок 11 - Створений рядок.  
Також є кнопка, щоб видалити параметри пошуку, що виведе на екран всі рядки цієї таблиці.

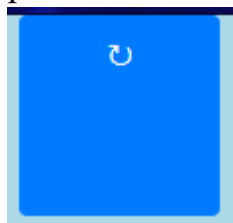


Рисунок 12 - Кнопка видалення пошуку.

Для виконання цієї графічної роботи, я використовував середовище розробки VS Code. Для підключення бази даних використовувалась платформа Node.js та додатково бібліотека “PG”.

Файл для підключення до БД.

db.js

```
const Pool = require('pg').Pool

//connecting to the local database
const pool = new Pool({
  user: "postgres",
  password: "1928sfsf",
  host: "localhost",
  port: 5432,
  database: "footballtournaments",
});

module.exports = pool;
```

Код програми написан мовою програмування JavaScript.