



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

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

**Лабораторна робота №2**  
з дисципліни  
«Бази даних і засоби управління»

Виконав студент III курсу  
ФПМ групи КВ-83  
Глеб В.Ю.  
Перевірів: Павловський В.І.

Київ – 2020

## **Ознайомлення з базовими операціями СУБД PostgreSQL**

*Завдання роботи полягає у наступному:*

1. Виконати нормалізацію бази даних, яка була створена у лабораторній роботі №1, до третьої нормальної форми (3НФ);
2. Реалізувати функціональні вимоги, наведені нижче.

*Функціональні вимоги*

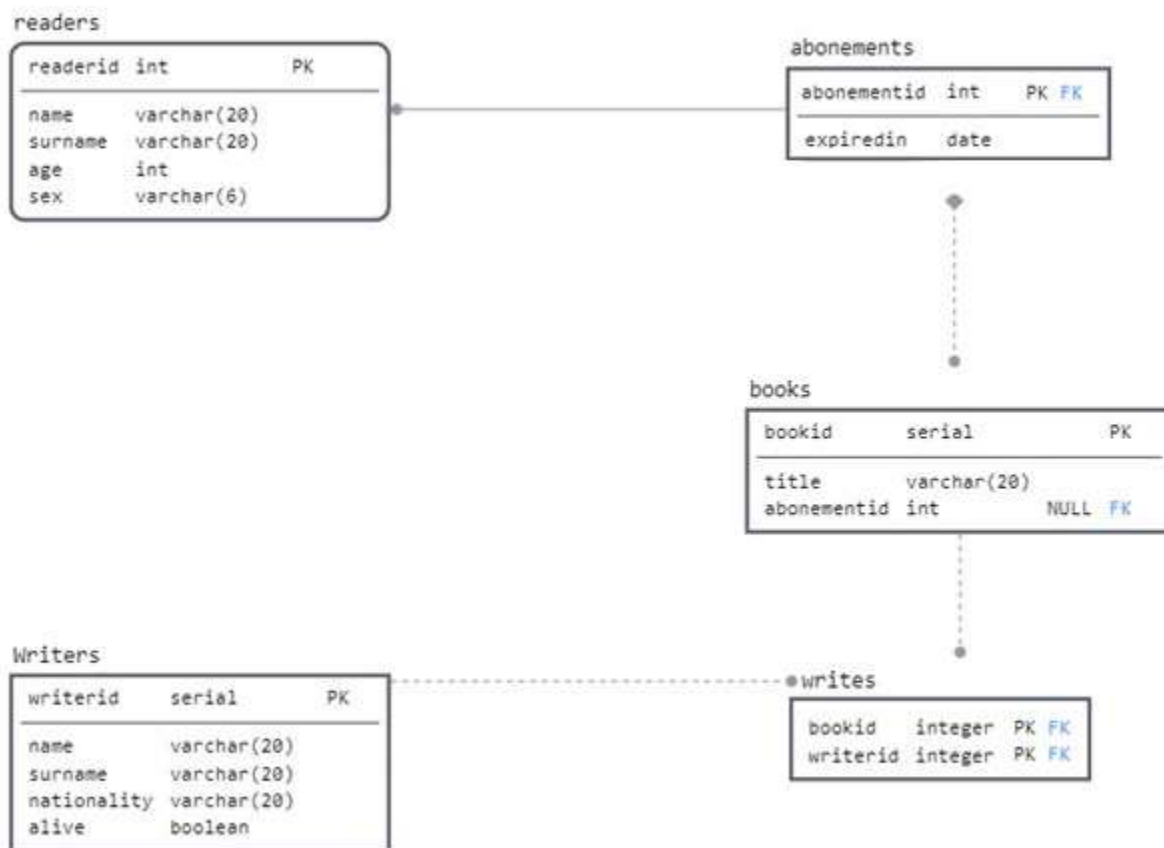
1. Реалізувати внесення, редагування та вилучення даних у базі засобами консольного інтерфейсу;
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі;
3. Забезпечити реалізацію пошуку за двома-трьома атрибутами з двох сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як перелічення, для логічного типу – значення True/False, для дат – у рамках діапазону дат;
4. Забезпечити реалізацію повнотекстового пошуку за будь-яким текстовим атрибутом бази даних засобами PostgreSQL з виділенням знайденого фрагменту.

*Вимоги до інтерфейсу користувача*

1. Використовувати консольний інтерфейс користувача.

### **Варіант 3**

**Нормалізована логічна модель даних БД «Бібліотека»**



**Рис 2.1 Нормалізована логічна модель даних БД «Бібліотека»**

Усі таблиці (відношення) знаходяться в 3 НФ тому, що у кожній із них:

1. Всі атрибути є атомарними та відсутні повторення рядків (1НФ);
2. Первинний ключ складається лише з одного атрибуту (2НФ);
3. Кожний не первинний атрибут не є транзитивно залежним від первинного ключа (3НФ).

## Опис програми

Програма створена для управління базою даних за допомогою базових операцій СУБД PostgreSQL та реалізовує функціональні вимоги, що наведені у завданні. Програма складається з 5 модулів:

1. `index.js` – точка входу до програми, запускає сервер, який приймає запити;
2. `router.js` – обробляє запити до сервера, підключає файл `controller.js`;
3. `controller.js` – обробляє логіку отримання даних. Підключає файл `tables.js`;
4. `tables.js` – містить класи, які відповідають за дані в таблицях БД. Для кожної таблиці створений свій клас;

## Структура меню програми

### Меню програми

The screenshot displays the user interface of a library management application. It features several floating forms for data entry and a table for viewing the library's inventory. The forms include:

- Add a reader:** Fields for Name, Surname, Age, and Sex.
- Add a book:** Fields for Book Title and Author.
- Add a books:** A form to add multiple books with a 'Number of books' field.
- Finder:** A search interface with a 'Test' field and a 'Get available books' button.
- Take a book:** Fields for Title and Reader.
- Remove a book:** A form to remove a book by ID.

Below the forms is a table titled "Library" with the following data:

BOOKID	TITLE	HOLDER	NATIONALITY	AUTHORS
3	Shining	Vladimir Landshut	Russian	Ian Tolstol
4	Hamlet	Book in library	UK	William Shakespeare
5	War and peace	Book in library	UK	William Shakespeare and Steven King
6	Hamlet	Book in library	American	Steven King
7	Hamlet	Book in library	Ukrainian	Taras Schorshchenko
10	Shining	Book in library	American	Steven King
11	Shining	Book in library	Russian	Ian Tolstol
12	Hamlet	Book in library	Ukrainian	Taras Schorshchenko
13	War and peace	Book in library	American	Steven King
14	Romeo and Juliet	Alex Stepota	American	Steven King
15	Romeo and Juliet	Book in library	UK	William Shakespeare
16	Shining	Book in library	American	Steven King
17	Romeo and Juliet	Book in library	Russian	Ian Tolstol
18	Hamlet	Book in library	American	Steven King
19	Hamlet	Book in library	Russian	Ian Tolstol
20	War and peace	Book in library	Ukrainian	Taras Schorshchenko

Рис 2.2 Меню програми

## Інтерфейс програми

1. Форма “Add a reader” дозволяє додати читача в бібліотеку. Передає на сервер (POST-запит) дані користувача. Дані зразу з’являються в інтерфейсі;
2. Форма “Add a book” дозволяє додати книжку в бібліотеку. Передає на сервер (POST-запит) з назвою книжки і іменем автора. Книжка з’являється в вікні “Library”;
3. Форма “Add books” дозволяє заповнити бібліотеку книжками. Передає на сервер (POST-запит) кількість необхідних книжок. Автори і назви беруться рандомно з вже існуючих книжок;
4. Форма “Finder” дозволяє шукати книжки, авторів, читачів(GET-запит):

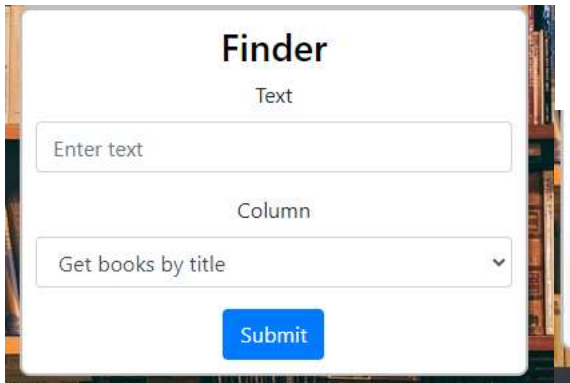


**Рис 2.3 Випадаюче вікно форми Finder**

- 4.1. “Get available books” виводить всі книжки, які знаходяться в бібліотеці
- 4.2. “Get readers” виводить дані всіх читачів;
- 4.3. “Get books by title” виводить всі книжки з введеною вище назвою;
- 4.4. “Get writers” виводить всіх авторів і інформацію про них;
5. Форма “Take a book” дозволяє взяти книгу з бібліотеки. Книга буде записана на читача, який був введений в формі.(PUT-запит);
6. Форма “Remove a book” видаляє книгу з бібліотеки.(DELETE-запит);
7. Таблиця “Library” містить актуальну інформацію про всі книжки і їх читачів.

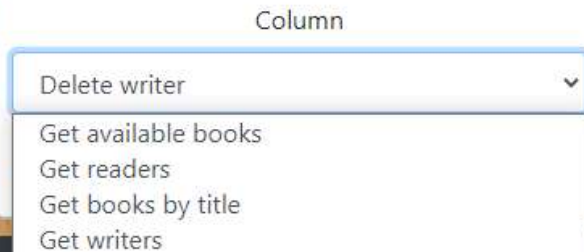
## Вибірка елементів з БД

В формі “Finder” реалізована вибірка елементів з різних таблиць і за різними полями:



The image shows a web form titled "Finder". It has a "Text" label above an input field with the placeholder "Enter text". Below this is a "Column" label above a dropdown menu currently showing "Get books by title". A blue "Submit" button is at the bottom.

Рис 2.4 Форма Finder



The image shows the dropdown menu for the "Column" field. The menu is open, showing the selected option "Delete writer" at the top, followed by "Get available books", "Get readers", "Get books by title", and "Get writers".

Рис 2.5 Випадаюче вікно форми Finder

1. “Get available books” – вибирає книги, в яких holder = null. При знаходженні співпадінь програма повертає масив книг з їх даними:



The image shows the Finder application window with a table titled "Library". The table has three columns: BOOKID, TITLE, and HOLDER. It contains 18 rows of data. The HOLDER column for all rows shows "Book in library".

BOOKID	TITLE	HOLDER
3	Shining	Book in library
4	Hamlet	Book in library
5	War and peace	Book in library
6	Hamlet	Book in library
7	Shining	Book in library
8	Shining	Book in library
9	Romeo and Juliet	Book in library
10	War and peace	Book in library
11	Romeo and Juliet	Book in library
12	Hamlet	Book in library
13	Romeo and Juliet	Book in library
14	Hamlet	Book in library
15	Shining	Book in library
16	Hamlet	Book in library
17	War and peace	Book in library
18	Romeo and Juliet	Book in library

Рис 2.6 Результат роботи “Get available books”

```

async availableBooks() {
  try {
    let res = await client.query('select * from books where holder is null')
    return res.rows
  } catch (e) {
    console.log(e)
    return false
  }
}

```

Рис 2.7 Метод, який виводить всі записи в яких `holder = null`

2. “Get readers” – виводить всіх читачів з бібліотеки:



The screenshot shows a web application window titled "Finder" with a close button in the top right corner. Inside the window, there is a table titled "Library". The table has six columns: "READERID", "NAME", "SURNAME", "AGE", "SEX", and "EXPIREDIN". There are seven rows of data, including one row with placeholder text "ExampleName" and "ExampleSurname".

READERID	NAME	SURNAME	AGE	SEX	EXPIREDIN
1	Vlad	Hleb	19	Male	12.12.2020
2	Yevgen	Levchuk	19	Male	12.05.2020
3	Ivan	Kucher	19	Male	12.07.2020
4	Nadia	Poluchovich	19	Female	15.12.2020
5	Alex	Nepota	19	Male	03.12.2020
6	Daria	Myagchova	19	Female	03.08.2020
7	ExampleName	ExampleSurname	18	Male	13.10.2021

Рис 2.8 Результат роботи “Get readers”

```

229   async finderReaders() {
230       try {
231           let res = await client.query('select readers.readerid,name,surname,age,sex,expiredin from readers
232           inner join abonements on abonements.abonementid = readers.readerid')
233           return res.rows
234       } catch (e) {
235           console.log(e)
236           return false
237       }
238   }

```

Рис 2.9 Метод, який виводить записи з таблиці “Readers”

3. “Get books by title” – вибирає книги з шуканою назвою:

BOOKID	TITLE	HOLDER
2	Romeo and Juliet	7
9	Romeo and Juliet	Book in library
11	Romeo and Juliet	Book in library
13	Romeo and Juliet	Book in library
18	Romeo and Juliet	Book in library
19	Romeo and Juliet	Book in library
26	Romeo and Juliet	Book in library
28	Romeo and Juliet	Book in library
29	Romeo and Juliet	Book in library
31	Romeo and Juliet	Book in library
32	Romeo and Juliet	Book in library
43	Romeo and Juliet	Book in library
52	Romeo and Juliet	Book in library
56	Romeo and Juliet	Book in library
61	Romeo and Juliet	Book in library
65	Romeo and Juliet	Book in library

Рис 2.9 Результат роботи методу “Get books by title”

```

257  async titleBooks() {
258      try {
259          let res = await client.query('select * from books where title = '${this.input}' order by bookid')
260          return res.rows
261      } catch (e) {
262          console.log(e)
263          return false
264      }
265  }
266  }

```

Рис 2.10 Метод, який виводить всі записи з таблиці “Books” з введеною в формі назвою

4. “Get writers” – виводить всіх письменників з бібліотеки:

WRITERID	NAME	SURNAME	NATIONALITY	ALIVE
1	Taras	Shevchenko	Ukrainian	false
2	Lesya	Ukrainka	Ukrainian	false
3	Lev	Tolstoi	Russian	false
4	Steven	King	American	true
5	William	Shakespeare	UK	false

Рис 2.11 Результат роботи “Get writers”



```

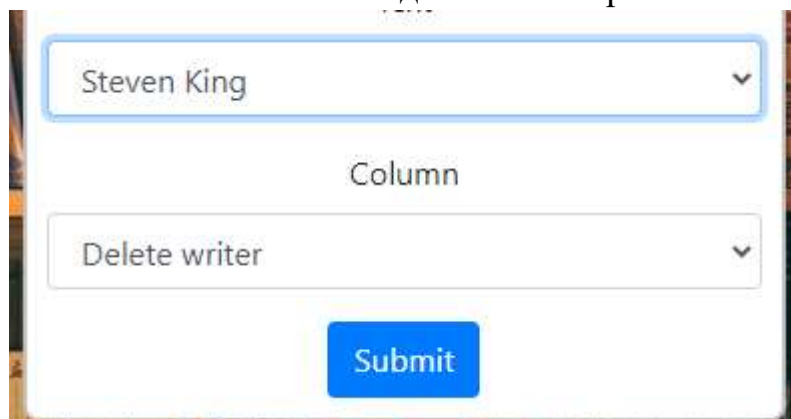
239  ✓      async finderWriters() {
240  ✓          try {
241              let res = await client.query('select * from writers')
242              return res.rows
243  ✓          } catch (e) {
244              console.log(e)
245              return false
246          }
247      }

```

**Рис 2.12 Метод, який виводить всі записи з таблиці “Writers”**

### **Видалення зв’язаних між собою даних.**

В програмі реалізована можливість видалення автора.



The image shows a web form with a light blue border. At the top, there is a dropdown menu with 'Steven King' selected. Below it is a label 'Column'. Under the label is another dropdown menu with 'Delete writer' selected. At the bottom of the form is a blue button with the text 'Submit'.

**Рис 2.13 Форма видалення Автора**

Оскільки таблиця Writers з’єднана з таблицею Writes і Books то видалення може бути здійснене тоді і тільки тоді, якщо в таблицях Writes і Books немає записів,

які містять ключі цього автора (ON DELETE NO ACTION). В випадку коли була здійснена спроба видалити автора програма видає застереження.

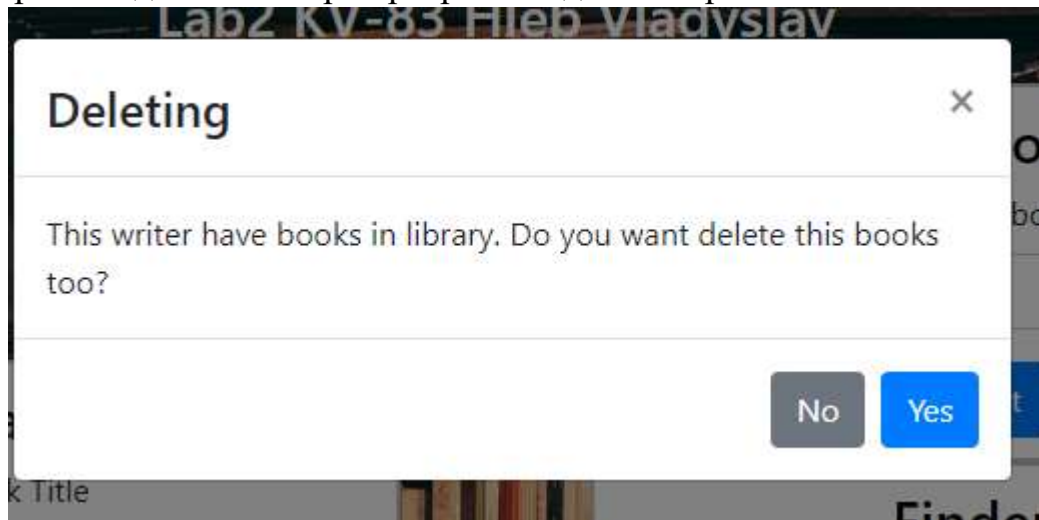


Рис 2.14 Підтвердження каскадного видалення

Якщо натиснути “Yes” програма змінить поле з NO ACTION на CASCADE і видалить всі дані про цього автора разом з книгами.

```
async deleteWriterClear() {
  try {
    let writerid = await client.query(`select writerid from writers where name = '${this.name}' and surname = '${this.surname}'`)
    writerid = writerid.rows[0].writerid
    await client.query(`
      alter table writes
      DROP constraint writerid;

      alter table writes add constraint writerid foreign key (writerid) REFERENCES writers(writerid) ON DELETE cascade;

      delete from writers where writerid = ${writerid};

      alter table writes
      DROP constraint writerid;
      alter table writes add constraint writerid foreign key (writerid) REFERENCES writers(writerid) ON DELETE no action;`)
    console.log('deleted')
    return true
  } catch (e) {
    console.log(e)
    console.log('troubles')
    return false
  }
}
```

Рис 2.15 Зміна режиму з NO ACTION на CASCADE

PostgreSQL також дає можливість вибрати кілька режимів видалення:

1. **“ON DELETE SET NULL”** - всі Foreign key будуть мати значення Null, а за неможливості цього зробити буде повідомлення про помилку.
2. **“ON DELETE RESTRICT”** – не дає можливості видалити батьківський рядок, якщо в нього є дочірні. Поведінка дуже схожа на NO ACTION

```

async deleteWriterClear(){
  try{
    let writerid = await client.query('select writerid from writers where name = '${this.name}' and surname = '${this.surname}');
    writerid = writerid.rows[0].writerid
    let writes = await client.query('select bookid from writes where writerid = ${writerid}')
    writes = writes.rows
    await client.query('delete from writes where writerid = ${writerid}')
    console.log('deleted')
    return true
  } catch(e){
    console.log(e)
    console.log('troubles')
    return false
  }
}

```

**Рис 2.16 Метод очистки всіх даних режимом “CASCADE”**

### Навігація програми

Програма створена за патерном MVC (Model-View-Controller). Складається з модулів Controller(index.js, router.js, controller.js), Model(books.js). Роль View виконує користувацький веб-інтерфейс. Опис файлів:

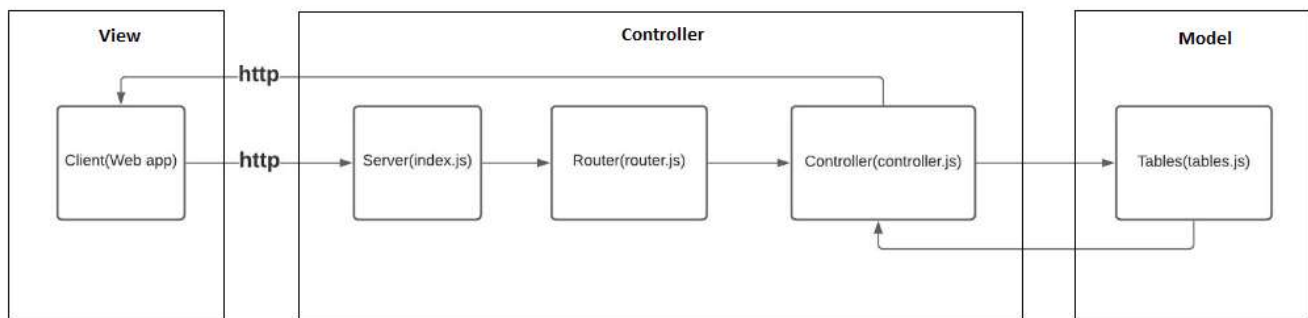
1. index.js (Рис 2.6) запускає сервер, підключає потрібні модулі і пакети
2. router.js (Рис 2.4) приймає запити з інтерфейсу користувача
3. controller.js(Рис 2.5) викликає потрібні методи для вибірки з БД
4. books.js (Рис 2.6) посилає запити в БД і віддає відповідь контролеру

### Посилання для навігації по програмі

1. [Index.js](#)
  - 1.1. [\(Line 25\) Запуск сервера](#)
2. [Router.js](#)
  - 2.1. [\(Line 5\) Роут, який приймає запит для отримання всіх існуючих книг з бібліотеки](#)
  - 2.2. [\(Line 8\) Роут, який приймає запит для отримання масиву авторів, книги яких є в бібліотеці](#)
  - 2.3. [\(Line 11\) Роут, який приймає запит для отримання масиву читачів, які зареєстровані в бібліотеці](#)
  - 2.4. [\(Line 14\) Роут, який приймає запит на отримання масиву книжок, які доступні для читання](#)
  - 2.5. [\(Line 17\) Роут, який приймає запит на пошук в бібліотеці за параметрами, описаними в тілі запиту](#)
  - 2.6. [\(Line 20\) Роут, який приймає запит на додавання користувача в бібліотеку](#)
  - 2.7. [\(Line 23\) Роут, який приймає запит на додавання певної кількості книг в бібліотеку. Кількість книг передається query-параметром.](#)
  - 2.8. [\(Line 26\) Роут, який приймає запит на видалення книги з бібліотеки. Ідентифікатор книги передається query-параметром](#)
  - 2.9. [\(Line 29\) Роут, який приймає запит на взяття книги з бібліотеки в користування читачем](#)
  - 2.10. [\(Line 32\) Роут, який приймає запит на додавання книги в бібліотеку](#)
3. [Controller.js](#)
  - 3.1. [\(Line 5\) Метод контролера, який бере дані з таблиці Books і відправляє відповідь клієнту](#)
  - 3.2. [\(Line 9\) Метод контролера, який бере дані з таблиці Readers і відправляє відповідь клієнту](#)
  - 3.3. [\(Line 13\) Метод контролера, який бере дані з таблиці Writers і відправляє відповідь клієнту](#)
  - 3.4. [\(Line 18\) Метод контролера, який бере дані з таблиці Books і відправляє відповідь клієнту](#)
  - 3.5. [\(Line 22\) Метод контролера, який шукає співпадіння в таблицях і відправляє відповідь клієнту](#)
  - 3.6. [\(Line 27\) Метод контролера, який додає запис в таблицю Readers](#)
  - 3.7. [\(Line 35\) Метод контролера, який додає запис в таблицю Books](#)

- 3.8. [\(Line 43\) Метод контролера, який видаляє запис з таблиці Books](#)
- 3.9. [\(Line 50\) Метод контролера, який змінює запис в таблиці Books](#)
- 3.10. [\(Line 56\) Метод контролера, який додає запис в таблицю Books](#)
4. [Tables.js](#)
  - 4.1. [\(Line 12\) Метод класу Books, який створює запис в таблиці Books](#)
  - 4.2. [\(Line 17\) Метод класу Books, який повертає ідентифікатор книги за назвою](#)
  - 4.3. [\(Line 26\) Метод класу Books, який видаляє книгу за ідентифікатором](#)
  - 4.4. [\(Line 35\) Метод класу Books, який повертає всі книги з бібліотеки](#)
  - 4.5. [\(Line 47\) Метод класу Books, який повертає всі назви книжок з таблиці Books](#)
  - 4.6. [\(Line 56\) Метод класу Books, який змінює запис в таблиці Books](#)
  - 4.7. [\(Line 65\) Метод класу Books, який створює n записів в таблиці](#)
  - 4.8. [\(Line 89\) Метод класу Writers, який повертає масив письменників](#)
  - 4.9. [\(Line 98\) Метод класу Writers, який повертає ідентифікатор потрібного письменника](#)
  - 4.10. [\(Line 116\) Метод класу Readers, який створює запис в таблиці Readers](#)
  - 4.11. [\(Line 131\) Метод класу Readers, який повертає всі записи в таблиці Readers](#)
  - 4.12. [\(Line 140\) Метод класу Readers, який повертає ідентифікатор читача](#)
  - 4.13. [\(Line 150\) Метод класу Readers, який повертає всі записи в таблиці Readers в рядковому типі](#)
  - 4.14. [\(Line 168\) Метод класу Finder, який перевіряє активні поля в запиті на пошук](#)
  - 4.15. [\(Line 197\) Метод класу Finder, який видаляє інформацію про письменника і всі його книги](#)
  - 4.16. [\(Line 219\) Метод класу Finder, який видаляє інформацію про письменника](#)
  - 4.17. [\(Line 229\) Метод класу Finder, який повертає всіх читачів](#)
  - 4.18. [\(Line 239\) Метод класу Finder, який повертає всіх письменників](#)
  - 4.19. [\(Line 248\) Метод класу Finder, який повертає всі книжки, в яких немає користувача](#)
  - 4.20. [\(Line 257\) Метод класу Finder, який шукає книги за назвою](#)
  - 4.21. [\(Line 272\) Метод класу Writes, який створює запис в таблиці Writes](#)

## Структура програми



**Рис 2.17 Структура програми**

### Додавання запису в таблицю “Books”

За додавання запису в таблицю Books відповідає форма “Add a book”.

Рис 2.18 Форма “Add a book”

В формі можна ввести назву книжки і вибрати автора з вже доступних в бібліотеці. При підтвердженні форми книга добавиться в таблицю “Books”:

id	title	book in library	language	author
105	War and peace	Book in library	American	Steven King
106	Example	Book in library	Ukrainian	Taras Schevchaenko

Рис 2.19 Результат роботи форми “Add a book”

### Метод, який додає книгу в бібліотеку

```

57   static async addBook(req, res) {
58       let book = new Books(req.body.title)
59       let bookid = await book.create()
60       let writerid = await Writers.getWriterId(req.body.writer)
61       let write = new Writes(writerid, bookid)
62       await write.create()
63       res.status(201).json()
64   }
65   }

```

```

class Books {
  constructor(title) {
    this.title = title
  }
  async create() {
    await client.query(`insert into books (title) values ('${this.title}')`)
    let bookid = await client.query('select bookid from books order by bookid desc limit 1')
    return bookid.rows[0].bookid
  }
}

```

Рис 2.20 Метод, який додає книгу в базу даних

### Додавання запису в таблицю “Readers”

За додавання запису в таблицю Readers відповідає форма “Add a reader”.



**Рис 2.21 Форма для додавання читача в таблицю “Readers”**

В формі можна ввести ім’я, фамілію, вік і стать.

При підтвердженні форми читач добавиться в таблицю “Readers”:

6	Daria	Myagchova	19	Female	03.08.2020
7	ExampleName	ExampleSurname	18	Male	13.10.2021

**Рис 2.22 Результат роботи додавання читача в базу даних**

```
109 class Readers {
110     constructor(name, surname, age, sex) {
111         this.name = name
112         this.surname = surname
113         this.age = age
114         this.sex = sex
115     }
116     async create() {
117         try {
118             await client.query('insert into readers (name,surname,age,sex) values ('${this.name}','${this.surname}', ${this.age}, '${this.sex}')')
119             let id = await client.query('select readerid from readers order by readerid desc limit 1')
120             console.log(id.rows[0].readerid)
121             await client.query('insert into abonements (abonementid, expiredin) values (${id.rows[0].readerid},
122             '${new Date((new Date().getFullYear() + 1).toString(),
123             new Date().getMonth(),
124             new Date().getDate()).toLocaleDateString()}')')
125             return true
126         } catch (e) {
127             console.log(e)
128             return false
129         }
130     }
131 }
```

**Рис 2.23 Методи, які додають читача в базу даних і створюють абонемент(tables.js 116-129)**

В разі помилки програма верне false (128 рядок).

### **Зміна запису в таблиці “Readers”**

За зміну запису в таблиці Readers відповідає форма “Take a book”.

В формі можна ввести книгу і читача з випадającego списку.

При підтвердженні форми в полі Holder появиться id читача , який взяв книгу:

**Рис 2.24 Форма “Take a book”**

BOOKID	TITLE	HOLDER	NATIONALITY	AUTHORS
1	Zapovit	Nadia Poluchovich	Ukrainian	Taras Shevchenko
2	Romeo and Juliet	ExampleName ExampleSurname	Ukrainian	Leiza Ukrainka

**Рис 2.24 Результат зміни запису, введеного в формі**

Метод, який змінює поле holder:

```

51  static async takeBook(req, res) {
52      let readerid = await Readers.getReaderId(req.body.holder)
53      let bookid = await Books.getBookId(req.body.title)
54      await Books.takeBook(bookid, readerid)
55      res.status(201).json()
56  }

56  static async takeBook(bookid, readerid) {
57      try {
58          await client.query(`update books set holder = ${readerid} where bookid = ${bookid}`)
59          return true
60      } catch (e) {
61          console.log(e)
62          return false
63      }
64  }

```

**Рис 2.25 Метод, який змінює поле holder**

## Додавання N записів в таблицю “Books”

За додавання записів відповідає Форма “Add books”



Рис 2.26 Форма Add books

В формі можна ввести потрібну кількість книг для додавання

BOOKID	TITLE	HOLDER	NATIONALITY	AUTHORS
1092	Hamlet	Book in library	UK	William Shakespeare
1093	Shining	Book in library	Ukrainian	Taras Shevchenko
1094	Hamlet	Book in library	Ukrainian	Taras Shevchenko
1095	Shining	Book in library	American	Steven King
1096	Shining	Book in library	Russian	Lev Tolstol
1097	Shining	Book in library	Russian	Lev Tolstol
1098	Romeo and Juliet	Book in library	Russian	Lev Tolstol
1099	Shining	Book in library	UK	William Shakespeare
1100	War and peace	Book in library	American	Steven King
1101	Shining	Book in library	American	Steven King
1102	Shining	Book in library	American	Steven King
1103	Romeo and Juliet	Book in library	American	Steven King
1104	Shining	Book in library	Ukrainian	Taras Shevchenko
1105	War and peace	Book in library	UK	William Shakespeare
1106	Hamlet	Book in library	UK	William Shakespeare
1107	Shining	Book in library	Ukrainian	Lena Libanska

Рис 2.27 Згенеровані книги на основі вже існуючих книг

```

65 static async addBooks(number) {
66     try {
67         let titles = await Books.getTitles()
68         console.log(titles)
69         let writers = await Writers.getWriters()
70         console.log(writers)
71         for (let i = 0; i < number; i++) {
72             let randomTitle = Math.floor(Math.random() * titles.length)
73             let randomWriter = Math.floor(Math.random() * writers.length)
74             let Book = new Books(titles[randomTitle])
75             let bookid = await Book.create()
76             let write = new Writes(await Writers.getWriterId(writers[randomWriter]), bookid)
77             await write.create()
78         }
79         return true
80     } catch (e) {
81         console.log(e)
82         return false
83     }
84 }
85

```

Рис 2.28 Метод, який додає книги



**Лістинг модуля index.js**

```

JS index.js > ...
1  const express = require('express')
2  const app = express()
3  const port = process.env.PORT || 8000
4  const bodyParser = require('body-parser')
5  const router = require('./controller/router/router')
6
7  app.use(bodyParser.json())
8  app.use(express.static(__dirname + '/build'));
9
10
11 app.get('/', (req, res) => {
12   res.sendFile(__dirname + '/build/index.html')
13 })
14 app.post('/api/addBook', router)
15 app.put('/api/takeBook', router)
16 app.post('/api/addBooks/:number', router)
17 app.delete('/api/deleteBook/:id', router)
18 app.post('/api/addReader', router)
19 app.get('/api/finder', router)
20 app.get('/api/getFreeTitles', router)
21 app.get('/api/getReaders', router)
22 app.get('/api/getWriters', router)
23 app.get('/api/getBooks', router)
24
25 const start = async () => {
26   app.listen(port, () => console.log("Server has been started"))
27 }
28
29 start()

```

Рис 2.29 Модуль index.js

```

controller > router > JS router.js > ...
1  const express = require('express')
2  const router = express.Router()
3  const Controller = require('../controller/controller')
4
5  router.route('/api/getBooks').get((req, res) => {
6      Controller.getBooks(req, res)
7  })
8  router.route('/api/getWriters').get((req, res) => {
9      Controller.getWriters(req, res)
10 })
11 router.route('/api/getReaders').get((req, res) => {
12     Controller.getReaders(req, res)
13 })
14 router.route('/api/getFreeTitles').get((req, res) => {
15     Controller.getFreeTitles(req, res)
16 })
17 router.route('/api/finder').get((req, res) => {
18     Controller.finder(req, res)
19 })
20 router.route('/api/addReader').post((req, res) => {
21     Controller.addReader(req, res)
22 })
23 router.route('/api/addBooks/:number').post(async (req, res) => {
24     Controller.addBooks(req, res)
25 })
26 router.route('/api/deleteBook/:id').delete((req, res) => {
27     Controller.deleteBook(req, res)
28 })
29 router.route('/api/takeBook').put((req, res) => {
30     Controller.takeBook(req, res)
31 })
32 router.route('/api/addBook').post((req, res) => {
33     Controller.addBook(req, res)
34 })
35
36 module.exports = router

```

Рис 2.30 Модуль router.js

```

controller > controller > js controller.js > ...
1  const { Books, Writers, Writes, Readers, Finder } = require('../model/tables')
2
3
4  class Controller {
5  static async getBooks(req, res) {
6      let response = await Books.getBooks()
7      res.status(200).json(response)
8  }
9  static async getReaders(req, res) {
10     let response = await Readers.getReadersArr()
11     res.status(200).json(response)
12 }
13 static async getWriters(req, res) {
14     let response = await Writers.getWriters()
15     res.status(200).json(response)
16 }
17 }
18 static async getFreeTitles(req, res) {
19     let response = await Books.getTitles()
20     res.status(200).json(response)
21 }
22 static async finder(req, res) {
23     const FR = new Finder(req.query.table, req.query.input)
24     let response = await FR.finder()
25     res.status(201).json(response)
26 }
27 static async addReader(req, res) {
28     let reader = new Readers(req.body.name, req.body.surname, req.body.age, req.body.sex)
29     if (await reader.create()) {
30         res.status(201).json()
31     } else {
32         res.status(403).json()
33     }
34 }
35 static async addBooks(req, res) {
36     let response = await Books.addBooks(req.params.number)
37     if (response) {
38         res.status(201).json()
39     } else {
40         res.status(403).json()
41     }
42 }
43 static async deleteBook(req, res) {
44     if (Books.deleteBook(req.params.id)) {
45         res.status(201).json()
46     } else {
47         res.status(403).json()
48     }
49 }

```

Рис 2.31\_1 Модуль controller.js

```

50 static async takeBook(req, res) {
51     let readerid = await Readers.getReaderId(req.body.holder)
52     let bookid = await Books.getBookId(req.body.title)
53     await Books.takeBook(bookid, readerid)
54     res.status(201).json()
55 }
56 static async addBook(req, res) {
57     let book = new Books(req.body.title)
58     let bookid = await book.create()
59     let writerid = await Writers.getWriterId(req.body.writer)
60     let write = new Writes(writerid, bookid)
61     await write.create()
62     res.status(201).json()
63 }
64 }
65
66
67
68
69
70 module.exports = Controller
71

```

Рис 2.31\_2 Модуль controller.js

### Лістинг модуля tables.js

```

model > JS tables.js > Books > getBooks
1  const { Client } = require('pg')
2  const client = new Client('postgres://postgres:vlad@localhost:5432/lab2')
3  client.connect().then(() => console.log('PG has been connected'))
4  const ReworkingData = require('./handlers')
5  const RD = new ReworkingData
6
7
8  class Books {
9      constructor(title) {
10         this.title = title
11     }
12     async create() {
13         await client.query(`insert into books (title) values ('${this.title}')`)
14         let bookid = await client.query('select bookid from books order by bookid desc limit 1')
15         return bookid.rows[0].bookid
16     }
17     static async getBookId(title) {
18         try {
19             let res = await client.query(`select bookid from books where title = '${title}'`)
20             return res.rows[0].bookid
21         } catch (e) {
22             console.log(e)
23             return false
24         }
25     }
26     static async deleteBook(id) {
27         try {
28             await client.query(`delete from books where bookid = ${id}`)
29             return true
30         } catch (e) {
31             console.log(e)
32             return false
33         }
34     }
35 }

```

Рис 2.32\_1 Модуль tables.js

```

35  ✓ static async getBooks() {
36  ✓     try {
37  ✓         let res = await client.query('select books.bookid,title,holder,name,surname,nationality from books
38             inner join writes on writes.bookid = books.bookid
39             inner join writers on writers.writerid = writes.writerid order by books.bookid')
40         let res2 = await Readers.getReaders()
41         return RD.reworkBooks(RD.handleBooks(res.rows, res2))
42  ✓     } catch (e) {
43         console.log(e)
44         return false
45     }
46 }
47  ✓ static async getTitles() {
48  ✓     try {
49         let res = await client.query('select title from books where holder is null')
50         return RD.reworkTitles(res.rows)
51  ✓     } catch (e) {
52         console.log(e)
53         return false
54     }
55 }
56  ✓ static async takeBook(bookid, readerid) {
57  ✓     try {
58         await client.query('update books set holder = ${readerid} where bookid = ${bookid}')
59         return true
60  ✓     } catch (e) {
61         console.log(e)
62         return false
63     }
64 }

65  static async addBooks(number) {
66      try {
67          let titles = await Books.getTitles()
68          console.log(titles)
69          let writers = await Writers.getWriters()
70          console.log(writers)
71          for (let i = 0; i < number; i++) {
72              let randomTitle = Math.floor(Math.random() * titles.length)
73              let randomWriter = Math.floor(Math.random() * writers.length)
74              let Book = new Books(titles[randomTitle])
75              let bookid = await Book.create()
76              let write = new Writes(await Writers.getWriterId(writers[randomWriter]), bookid)
77              await write.create()
78          }
79          return true
80      } catch (e) {
81          console.log(e)
82          return false
83      }
84  }
85
86  }
87 }

```

Рис 2.32\_2 Модуль tables.js

```

88 class Writers {
89     static async getWriters() {
90         try {
91             let res = await client.query('select name, surname from writers')
92             return RD.reworkPeople(res.rows)
93         } catch (e) {
94             console.log(e)
95             return false
96         }
97     }
98     static async getWriterId(writerString) {
99         let writer = writerString.split(" ")
100         try {
101             let res = await client.query(`select writerid from writers where name = '${writer[0]}' and surname = '${writer[1]}'`)
102             return res.rows[0].writerid
103         } catch (e) {
104             console.log(e)
105             return false
106         }
107     }
108 }

109 class Readers {
110     constructor(name, surname, age, sex) {
111         this.name = name
112         this.surname = surname
113         this.age = age
114         this.sex = sex
115     }
116     async create() {
117         try {
118             await client.query(`insert into readers (name,surname,age,sex) values ('${this.name}','${this.surname}', ${this.age}, '${this.sex}'))`)
119             let id = await client.query('select readerid from readers order by readerid desc limit 1')
120             console.log(id.rows[0].readerid)
121             await client.query(`insert into abonements (abonementid, expiredin) values (${id.rows[0].readerid},
122             '${new Date((new Date().getFullYear() + 1).toString(),
123             new Date().getMonth(),
124             new Date().getDate()).toLocaleDateString()}')`)
125             return true
126         } catch (e) {
127             console.log(e)
128             return false
129         }
130     }
131     static async getReaders() {
132         try {
133             let res = await client.query('select * from readers')
134             return res.rows
135         } catch (e) {
136             console.log(e)
137             return false
138         }
139     }
140     static async getReaderId(fio) {
141         let reader = fio.split(" ")
142         try {
143             let res = await client.query(`select readerid from readers where name = '${reader[0]}' and surname = '${reader[1]}'`)
144             return res.rows[0].readerid
145         } catch (e) {
146             console.log(e)
147             return false
148         }
149     }

150     static async getReadersArr() {
151         try {
152             let res = await client.query('select * from readers')
153             return RD.reworkPeople(res.rows)
154         } catch (e) {
155             console.log(e)
156             return false
157         }
158     }
159 }

```

Рис 2.32\_3 Модуль tables.js

```

158 class Finder {
159   constructor(table, input, name, surname, clear) {
160     this.table = table
161     this.input = input
162     this.name = name
163     this.surname = surname
164     this.clear = clear
165   }
166   async finder() {
167     console.log(this.table, this.input, this.name, this.surname, this.clear)
168     switch (this.table) {
169       case 'Readers': {
170         return await this.finderReaders()
171       }
172       case 'Books': {
173         if (this.input === '') {
174           return await this.availableBooks()
175         } else {
176           return await this.titleBooks()
177         }
178       }
179       case 'Writers': {
180         if (this.name && this.surname) {
181           if (this.clear === 'true') {
182             return await this.deleteWriterClear()
183           } else {
184             console.log(2)
185             return await this.deleteWriter()
186           }
187         } else {
188           return await this.finderWriters()
189         }
190       }
191     }
192   }
193   async deleteWriterClear(){
194     try{
195       let writerid = await client.query('select writerid from writers where name = '${this.name}' and surname = '${this.surname}')
196       writerid = writerid.rows[0].writerid
197       let writes = await client.query('select bookid from writes where writerid = ${writerid}')
198       writes = writes.rows
199       writes.map((elem) => {
200         client.query('delete from books where bookid = ${elem.bookid}')
201       })
202       writes.map((elem) => {
203         client.query('delete from writes where writerid = ${writerid}')
204       })
205       await client.query('delete from writers where writerid = ${writerid}')
206       console.log('deleted')
207       return true
208     } catch(e){
209       console.log(e)
210       console.log('troubles')
211       return false
212     }
213   }
214 }
215

```

Рис 2.32\_4 Модуль tables.js



```

219     async deleteWriter(){
220         try{
221             await client.query(`delete from writers where name = '${this.name}' and surname = '${this.surname}'`)
222             return true
223         }catch(e){
224             console.log(e)
225             return false
226         }
227     }
228 }
229 async finderReaders() {
230     try {
231         let res = await client.query(`select readers.readerid,name,surname,age,sex,expiredin from readers
232         inner join abonements on abonements.abonementid = readers.readerid`)
233         return res.rows
234     } catch (e) {
235         console.log(e)
236         return false
237     }
238 }
239 async finderWriters() {
240     try {
241         let res = await client.query('select * from writers')
242         return res.rows
243     } catch (e) {
244         console.log(e)
245         return false
246     }
247 }
248 async availableBooks() {
249     try {
250         let res = await client.query('select * from books where holder is null')
251         return res.rows
252     } catch (e) {
253         console.log(e)
254         return false
255     }
256 }
257 async titleBooks() {
258     try {
259         let res = await client.query(`select * from books where title = '${this.input}' order by bookid`)
260         return res.rows
261     } catch (e) {
262         console.log(e)
263         return false
264     }
265 }
266 }

```

Рис 2.32\_5 Модуль tables.js