

**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»**

Тема: «Створення додатку бази даних, орієнтованого на взаємодію з СУБД  
PostgreSQL»

Виконав: студент III курсу

ФПМ групи КВ-94

Жила Д.М.

Перевірів: доц. Петрашенко А. В.

Київ – 2021

*Мета роботи:* здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

*Загальне завдання роботи полягає у наступному:*

1. Реалізувати функції внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з 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).

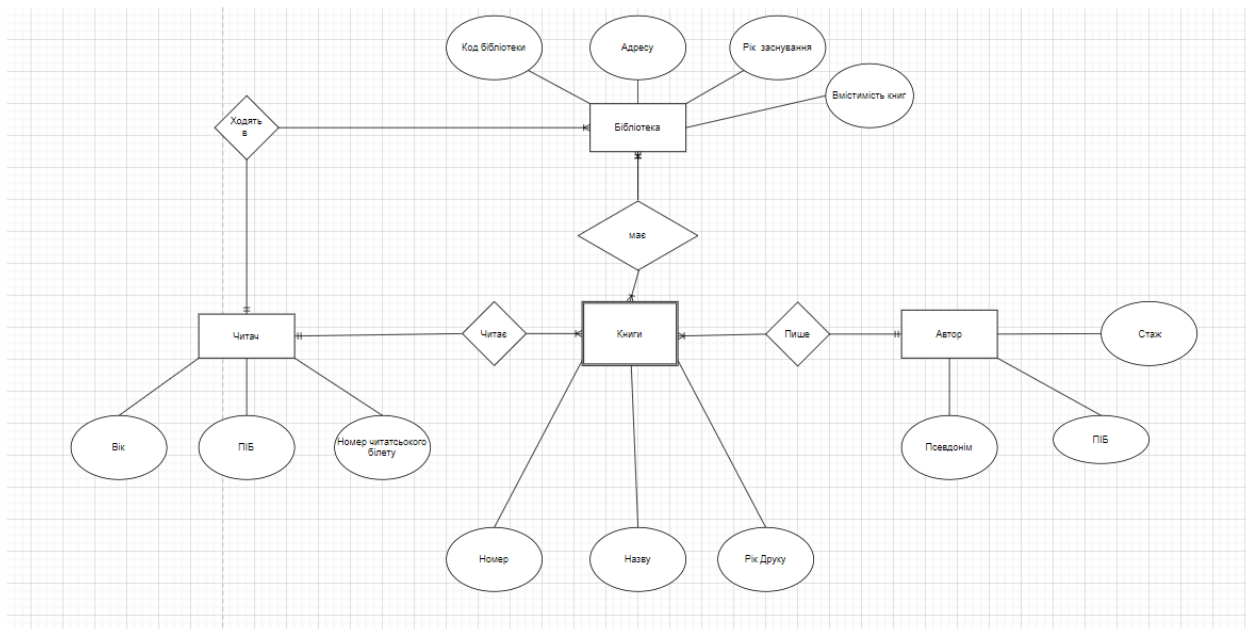
*Деталізоване завдання:*

1. Забезпечити можливість введення/редагування/вилучення даних у таблицях бази даних з можливістю контролю відповідності типів даних атрибутів таблиць (рядків, чисел, дати/часу). Для контролю пропонується два варіанти: контроль при введенні (валідація даних) та перехоплення помилок (try..except) від сервера PostgreSQL при виконанні відповідної команди SQL. Особливу увагу варто звернути на дані таблиць, що мають зв'язок 1:N. При цьому з боку батьківської таблиці необхідно контролювати **вилучення** рядків за умови наявності даних у підлеглий таблиці. З точки зору підлеглої таблиці варто контролювати наявність відповідності рядка у батьківській таблиці при виконанні **внесення** нових даних. Унеможливити виведення програмою системних помилок на екрані шляхом їх перехоплення і адекватної обробки. Внесення даних виконується у консольному вікні програми.
2. Забезпечити можливість автоматичної генерації великої кількості даних у таблицях за допомогою вбудованих у PostgreSQL функцій роботи з псевдовипадковими числами. Дані мають бути згенерованими **не мовою програмування, а відповідним SQL-запитом!**
3. Для реалізації пошуку необхідно підготувати 3 запити, включають дані з декількох таблиць і фільтрують рядки за 3-4 атрибутами цих таблиць. Забезпечити можливість введення конкретних значень констант для фільтрації з клавіатури користувачем. Крім того, після виведення даних необхідно вивести час виконання запиту у мілісекундах. Перевірити швидкодію роботи запитів на попередньо згенерованих даних.

- Програмний код організувати згідно шаблону Model-View-Controller(MVC). При цьому модель, подання та контролер мають бути реалізовані у окремих файлах. Для доступу до бази даних використовувати **лише мову SQL** (без ORM).

### Інформація про модель та структуру бази даних

Рис. 1 - Концептуальна модель предметної області “Облік книгозбірні”



Нижче (Рис. 2) наведено логічну модель бази даних:

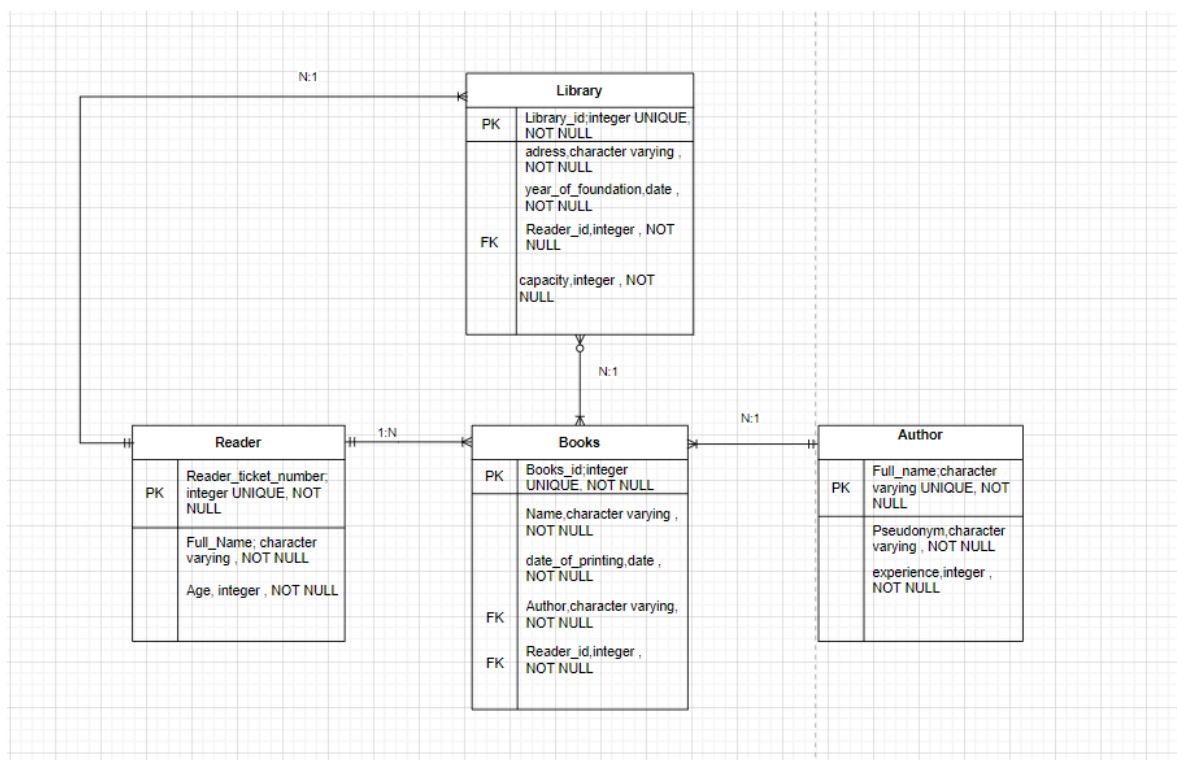


Рис. 2 – Логічна модель бази даних

Зміни у порівнянні з першою лабораторною роботою відсутні.

## Середовище розробки та налаштування підключення до бази даних

Для виконання лабораторної роботи використовувалась мова програмування Python та текстовий редактор Sublime Text 3.

Для підключення до серверу бази даних PostgreSQL використано модуль «psycopg2».

## Опис структури програми

Програма містить 5 основних модулів: **Lab, model, view, controller, utils**.  
Файл для запуску - «lab2.py3».

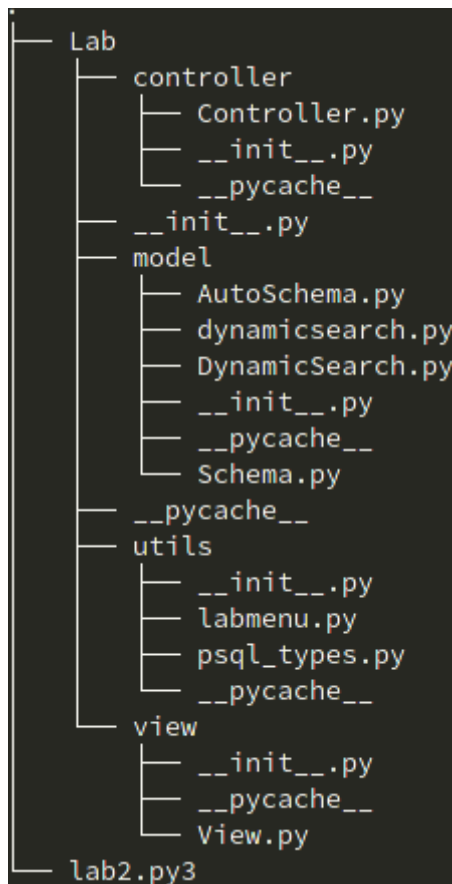


Рис. 3 – Структура програмного коду

## Структура меню програми

Головне меню

```
MVC schema "Library" interface
> "Author" table
  "Books" table
  "Reader" table
  "Library" table
  "LibraryBooks" table
  Schema "Library" utils
  Dynamic search
  exit
```

Меню для таблиці

```
"Library"."Author" table interface:
> describe
  show data
  add data
  edit data
  remove data
  random fill
  return
```

Меню для вибору динамічних запитів

```
Schema "Library" dynamic search interface
> Books
  ReaderLoan
  Reader
  return

Books dynamic search interface
"id" ignored
"date_of_printing" ignored
"Name" ignored
"Full_name" ignored
"Pseudonym" ignored
"experience" ignored
> id
  date_of_printing
  Name
  Full_name
  Pseudonym
  experience
  execute
  sql
  reset
  return
```

Меню вибору кількості рядків для генерації

```
instances [100]:
```

### Пункт 1

Реалізувати функції внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.

### Внесення даних

Створення нового Автора:

```
id | Full_name | Pseudonym | experience
0 rows, execution time: 0:00:00.000552
```

```
"Library"."Author" table interface:
describe
show data
> add data
edit data
remove data
random fill
return
```

```
Full_name: Dmytro ZHyla
Pseudonym: dedmon
experience: 666
1 rows added
```

```
id | Full_name | Pseudonym | experience
1 | Dmytro ZHyla | dedmon | 666
1 rows, execution time: 0:00:00.000489
```

### Видалення даних

```
id | Full_name | Pseudonym | experience
1 | Dmytro ZHyla | dedmon | 666
1 rows, execution time: 0:00:00.000489
```

```
"Library"."Author" table interface:
describe
show data
add data
edit data
> remove data
random fill
return
```

```
id: 1
1 rows deleted
```

```
id | Full_name | Pseudonym | experience
0 rows, execution time: 0:00:00.000592
```

Якщо уведено неіснуючий id рядку:

```
id: 333
0 rows deleted
```

Неправильно введене число:

```
id: a33
Error: 'a33' is not a valid integer.
id: 
```

При видаленні рядку програма завжди використовує ключове слово “CASCADE” мови SQL. Ключове слово “CASCADE” дає дозвіл СУБД автоматично видаляти залежні рядки в дочірній таблиці, коли відповідні рядки видаляються в батьківській таблиці.

### Редагування даних

```
id | Full_name | Pseudonym | experience
2  | Dmytri Zhyka | darmi | 9
1 rows, execution time: 0:00:00.000264
```

**"Library"."Author" table interface:**

```
describe
show data
add data
> edit data
remove data
random fill
return
```

```
id: 2
Full_name: Daniel Defo
Pseudonym: dan4ik
experience: 33
1 rows changed
```

```
id | Full_name | Pseudonym | experience
2  | Daniel Defo | dan4ik | 33
1 rows, execution time: 0:00:00.000302
```

## Пункт 2

Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.

У програмі передбачено рандомізоване заповнення кожної таблиці окремо(з вказаннями кількості рандомізованих рядків для генерації), та пакетне рандомізоване заповнення таблиць схеми(без можливості зміни кількості рандомізованих рядків користувачем).

### Рандомізоване заповнення таблиці “Author”:

```
id | Full_name | Pseudonym | experience
0 rows, execution time: 0:00:00.000463
"Library"."Author" table interface:
> describe
show data
add data
edit data
remove data
random fill
return
```

```
instances [100]: 5
"Library"."Author" 5 rows added, execution time: 0:00:00.000999
```

```
id | Full_name | Pseudonym | experience
3 | BNM | DFGHJKLZXC | 99
4 | uiopasdfgh | DFGHJKLZXC | 7
5 | ertyuiopas | zxcvbnmQWE | 79
6 | OPASDFGHJK | ASDFGHJKLZ | 72
7 | ZXCVBNM | RTYUIOPASD | 43
5 rows, execution time: 0:00:00.000519
```

### Пакетне заповнення таблиць схеми:

Кількість заданих рядків пакетного заповнення таблиць схеми:

Author	1000
Reader	1000
Books	1000
Library	1000
LibraryBooks	1000

```
"Library"."Author" 1000 rows added, execution time: 0:00:00.008848
"Library"."Reader" 1000 rows added, execution time: 0:00:00.007579
"Library"."Books" 1000 rows added, execution time: 0:00:01.151539
"Library"."Library" 1000 rows added, execution time: 0:00:00.579413
"Library"."LibraryBooks" 1000 rows added, execution time: 0:00:01.194470
```



Витяги деяких рандомізованих рядків з таблиць:

### Author

id	Full_name	Pseudonym	experience
3	BNM	DFGHJKLZXC	99
4	uiopasdfgh	DFGHJKLZXC	7
5	ertyuiopas	zxcvbnmQWE	79
6	OPASDFGHJK	ASDFGHJKLZ	72
7	ZXCVBNM	RTYUIOPASD	43
8	uiopasdfgh	WERTYUIOPA	19
9	bnmQWERTYU	bnmQWERTYU	60
10	nmQWERTYUI	RTYUIOPASD	78
11	bnmQWERTYU	OPASDFGHJK	5
12	DFGHJKLZXC	asdfghjklz	94
13	vbnmQWERTY	QWERTYUIOP	88
14	HJKLZXCVCBN	BNM	22
15	sdfghjklzx	ASDFGHJKLZ	60
16	mQWERTYUIO	ZXCVBNM	5
17	tyuiopasdf	iopasdfghj	86
18	TYUIOPASDF	cvbnmQWERT	70
19	qwertyuiop	qwertyuiop	81
20	uiopasdfgh	RTYUIOPASD	40
21	ertyuiopas	mQWERTYUIO	65
22	YUIOPASDFG	SDFGHJKLZX	88
23	PASDFGHJKL	tyuiopasdf	85
24	l zxcvbnmQW	opasdfghik	4

### Reader

id	Full_Name	Age
1	FGHJKLZXC	28
2	QWERTYUIOP	35
3	tyuiopasdf	29
4	tyuiopasdf	3
5	FGHJKLZXC	21
6	dfghjklzxc	58
7	QWERTYUIOP	29
8	DFGHJKLZXC	44
9	PASDFGHJKL	20
10	UIOPASDFGH	76
11	M	93
12	pasdfghjkl	39
13	cvbnmQWERT	18
14	rtyuiopasd	95
15	yuiopasdfg	37
16	VBNM	18

### Books

id	date_of_printing		Name	Author	Reader_id
1	2021-11-05	21:19:25.227196+02:00	iopasdfghj	32	306
2	2021-04-27	02:38:57.069868+03:00	hijklzxcvbn	9	812
3	2021-02-28	01:04:59.030415+02:00	ASDFGHJKLZ	220	27
4	2021-07-20	19:04:17.964075+03:00	ZXCVBNM	92	460
5	2021-06-14	15:20:39.728101+03:00	asdfghjklz	823	364
6	2021-08-08	18:16:03.588529+03:00	sdfghjklzx	391	346
7	2021-07-23	13:27:06.769501+03:00	QWERTYUIOP	489	394
8	2021-10-12	21:45:17.226079+03:00	DFGHJKLZXC	292	827
9	2021-05-08	15:50:09.914246+03:00	sdfghjklzx	102	157
10	2021-06-17	05:24:56.988409+03:00	ertyuiopas	691	177
11	2021-03-17	04:20:39.551548+02:00	tyuiopasdf	406	837
12	2021-03-05	08:25:19.843354+02:00	SDFGHJKLZX	974	558
13	2021-02-28	03:06:10.356496+02:00	zxcvbnmQWE	605	871
14	2021-09-11	04:17:17.679644+03:00	sdfghjklzx	412	630
15	2021-08-03	04:41:16.423235+03:00		37	671
16	2021-10-14	13:30:31.963850+03:00	jklzxcvbnm	313	609
17	2021-05-21	05:12:54.742498+03:00	xcvbnmQWER	671	670

## Library

id	year_of_foundation	address	capacity	Reader_id
1	2021-04-15 12:21:07.373052+03:00	vbnmQWERTY	95	289
2	2021-05-03 11:12:10.505567+03:00	ertyuiopas	51	20
3	2021-06-12 04:14:22.464649+03:00	FGHJKLZXCV	75	32
4	2021-07-19 02:59:16.444223+03:00	bnmQWERTYU	34	180
5	2021-02-08 13:43:55.225713+02:00	jklzxcvbnm	52	936
6	2021-06-23 19:41:26.645174+03:00	XCVBNM	74	423
7	2021-05-16 15:21:03.668700+03:00	SDFGHJKLZX	92	558
8	2021-09-12 18:46:23.912062+03:00	LZXCVBNM	9	412
9	2021-11-08 23:28:57.651421+02:00	JKLZXCVBNM	33	296
10	2021-10-28 04:42:53.599229+03:00	ghjklzxcvb	84	125
11	2021-10-31 06:20:40.898065+02:00	QWERTYUIOP	34	856
12	2021-04-25 14:51:54.713922+03:00	FGHJKLZXCV	16	759
13	2021-05-10 04:52:00.750083+03:00	uiopasdfgh	1	368
14	2021-05-17 20:24:12.891557+03:00	mQWERTYUIO	97	445
15	2021-02-15 21:45:23.065892+02:00	vbnmQWERTY	8	441
16	2021-08-28 05:38:01.038367+03:00	PASDFGHJKL	43	751
17	2021-08-10 15:37:56.301813+03:00	XCVBNM	63	874
18	2021-05-09 10:32:17.143557+03:00	zxcvbnmQWE	71	13
19	2021-01-24 17:57:41.676336+02:00	HJKLZXCVBN	29	156
20	2021-08-06 04:39:26.264354+03:00	qwertyuiop	97	140
21	2021-06-29 09:39:25.132060+03:00	VBNM	89	503
22	2021-03-04 19:52:15.972104+02:00	tyuiopasdf	85	214

## LibraryBooks

id	Library_id	Book_id
1	13	985
2	987	27
3	443	650
4	699	43
5	491	335
6	510	258
7	92	423
8	47	241
9	711	934
10	528	778
11	212	776
12	452	287
13	688	498
14	381	310
15	143	477
16	350	807
17	621	493
18	834	849
19	26	408
20	577	524
21	979	655
22	34	983

## SQL запити рандомізованого заповнення:

```
lab3 INSERT INTO "Library"."Author"("Full_name", "Pseudonym", "experience")
SELECT
    substr(characters, (random() * length(characters) + 1)::integer, 10),
    substr(characters, (random() * length(characters) + 1)::integer, 10),
    trunc(random() * 100)::int
FROM
    (VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters),
    generate_series(1, 1000) as q;

"Library"."Author" 1000 rows added, execution time: 0:00:00.010122
```

```
trunc(random() * 100)::int INSERT INTO "Library"."Reader"("Full_Name", "Age")
SELECT
    substr(characters, (random() * length(characters) + 1)::integer, 10),
    trunc(random() * 100)::int
FROM
    (VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters),
    generate_series(1, 1000) as q;

"Library"."Reader" 1000 rows added, execution time: 0:00:00.007584
```

```
lab3 INSERT INTO "Library"."Books"("date_of_printing", "Name", "Author", "Reader_id")
SELECT
    timestamp '2021-01-01' + random() * (timestamp '2021-11-11' - timestamp '2021-01-01'),
    substr(characters, (random() * length(characters) + 1)::integer, 10),
    (SELECT "id" FROM "Library"."Author" ORDER BY random()*q LIMIT 1),
    (SELECT "id" FROM "Library"."Reader" ORDER BY random()*q LIMIT 1)
FROM
    (VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters),
    generate_series(1, 1000) as q;

^[[A^[[A^[[A^[[B"Library"."Books" 1000 rows added, execution time: 0:00:02.017129
```

```
lab2 INSERT INTO "Library"."Library"("year_of_foundation", "address", "capacity", "Reader_id")
SELECT
    timestamp '2021-01-01' + random() * (timestamp '2021-11-11' - timestamp '2021-01-01'),
    substr(characters, (random() * length(characters) + 1)::integer, 10),
    trunc(random() * 100)::int,
    (SELECT "id" FROM "Library"."Reader" ORDER BY random()*q LIMIT 1)
FROM
    (VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters),
    generate_series(1, 1000) as q;

^["Library"."Library" 1000 rows added, execution time: 0:00:01.070699
```

```
INSERT INTO "Library"."LibraryBooks"("Library_id", "Book_id")
SELECT
    (SELECT "id" FROM "Library"."Library" ORDER BY random()*q LIMIT 1),
    (SELECT "id" FROM "Library"."Books" ORDER BY random()*q LIMIT 1)
FROM
    (VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters),
    generate_series(1, 1000) as q;

"Library"."LibraryBooks" 1000 rows added, execution time: 0:00:01.927923
```

### Пункт 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.



Було підготовлено два SQL запити:

- Пошук читача за атрибутами:
  - Повне ім'я читача
  - Кількість повних років читача
  - ID читача в бібліотеці
- Пошук книги за атрибутами
  - ID книги
  - Дата коли було видано книгу
  - Назва книги

Пошук читача:

SQL запит без фільтрації рядків:

```
SELECT
    "a"."id" as "id",
    "a"."date_of_printing" as "date_of_printing",
    "a"."Name" as "Name",
    -- "b"."Full_name" as "Full_name",
    -- "b"."Pseudonym" as "Pseudonym",
    -- "b"."experience" as "experience"
FROM
    "Library"."Books" as "a"
    -- INNER JOIN "Library"."Author" as "b"
    --      ON "a"."Author" = "b"."id"
;
```

Результат:

id	Full_name	Pseudonym	experience
3	BNM	DFGHJKLZXC	99
4	uiopasdfgh	DFGHJKLZXC	7
5	ertyuiopas	zxcvbnmQWE	79
6	OPASDFGHJK	ASDFGHJKLZ	72
7	ZXCVBNM	RTYUIOPASD	43
8	uiopasdfgh	WERTYUIOPA	19
9	bnmQWERTYU	bnmQWERTYU	60
10	nmQWERTYUI	RTYUIOPASD	78
11	bnmQWERTYU	OPASDFGHJK	5
12	DFGHJKLZXC	asdfghjklz	94
13	vbnmQWERTY	QWERTYUIOP	88
14	HJKLZXCVBN	BNM	22
15	sdfghjklzx	ASDFGHJKLZ	60
16	mQWERTYUIO	ZXCVBNM	5
17	tyuiopasdf	iopasdfghj	86

Налаштування фільтрування рядків:

- ID читача  $\geq 66$
- ім'я читача LIKE 'b%'
- Вік читача  $\leq 30$

Введені критерії фільтрування:

```
Reader dynamic search interface
"id" >= 66::bigint
"Full_Name" LIKE '%b'::varchar
"Age" <= 30::bigint
```

SQL запит з заданими налаштуваннями фільтрування рядків:

```
SELECT
    "a"."id" as "id",
    "a"."Full_Name" as "Full_Name",
    "a"."Age" as "Age"
    -- "b"."id" as "id",
    -- "b"."year_of_foundation" as "year_of_foundation",
    -- "b"."address" as "address",
    -- "b"."capacity" as "capacity"
FROM
    "Library"."Reader" as "a"
    -- INNER JOIN "Library"."Library" as "b"
    -- ON "a"."id" = "b"."Reader_id"
WHERE
    ("a"."id" >= 66::bigint) AND
    ("a"."Full_Name" LIKE '%b'::varchar) AND
    ("a"."Age" <= 30::bigint);
```

Результат:

id	Full_Name	Age
137	ghjklzxcvb	6
412	ghjklzxcvb	21
617	ghjklzxcvb	5
832	ghjklzxcvb	27
1090	ghjklzxcvb	15
1099	ghjklzxcvb	2
1209	ghjklzxcvb	29
1318	ghjklzxcvb	1
1365	ghjklzxcvb	22
1441	ghjklzxcvb	14
1656	ghjklzxcvb	27
1678	ghjklzxcvb	1
1787	ghjklzxcvb	10
1822	ghjklzxcvb	26

14 rows, execution time: 0:00:00.001382

### SQL запит без фільтрації рядків:

```
SELECT
    "a"."id" as "id",
    "a"."date_of_printing" as "date_of_printing",
    "a"."Name" as "Name"

    -- "b"."Full_name" as "Full_name",
    -- "b"."Pseudonym" as "Pseudonym",
    -- "b"."experience" as "experience"

FROM
    "Library"."Books" as "a"
    -- INNER JOIN "Library"."Author" as "b"
    --     ON "a"."Author" = "b"."id"
;
```

```

1982 | 2021-02-15 01:43:20.088031+02:00 | ertyuiopasd
1983 | 2021-09-08 06:40:22.012017+03:00 | fghjklzxcv
1984 | 2021-02-27 22:14:34.235290+02:00 | FGHJKLZXCV
1985 | 2021-05-24 05:31:42.163317+03:00 | HJKLZXCVBN
1986 | 2021-05-03 09:16:43.649661+03:00 | fghjklzxcv
1987 | 2021-08-08 16:11:21.583065+03:00 | rtyuiopasd
1988 | 2021-06-14 12:24:12.625272+03:00 | DFGHJKLZXC
1989 | 2021-04-30 07:49:33.561294+03:00 | HJKLZXCVBN
1990 | 2021-02-26 22:43:41.703772+02:00 | hjklzxcvbn
1991 | 2021-07-25 13:01:28.363978+03:00 | XCVBNM
1992 | 2021-09-10 18:58:01.299648+03:00 | uiopasdfgh
1993 | 2021-09-02 16:28:04.721279+03:00 | dfghjklzxc
1994 | 2021-01-08 00:51:07.205396+02:00 | wertyuiopa
1995 | 2021-11-07 12:50:04.497677+02:00 | klzxcvbnmQ
1996 | 2021-05-08 10:20:26.519481+03:00 | bnmQWERTYU
1997 | 2021-03-06 08:23:33.170330+02:00 | qwertyuiop
1998 | 2021-02-15 03:40:32.351730+02:00 | lzxcvbnmQW
1999 | 2021-10-23 15:09:27.324701+03:00 | OPASDFGHJK
2000 | 2021-06-07 00:00:12.131689+03:00 | M
2000 rows, execution time: 0:00:00.002586

```

Налаштування фільтрування рядків:

- За ID кинги < 77
- За датою видачі книги >= 2001-04-09
- За назвою книги LIKE `NM`

SQL запит з заданими налаштуваннями фільтрування рядків:

Введені критерії фільтрування:

```
Books dynamic search interface
"id" < 77::bigint
"date_of_printing" >= '2001-04-09 00:00:00'::timestamp
"Name" LIKE 'NM'::varchar
```

SQL запит з заданими налаштуваннями фільтрування рядків:

```
SELECT
  "a"."id" as "id",
  "a"."date_of_printing" as "date_of_printing",
  "a"."Name" as "Name",
  "b"."Full_name" as "Full_name",
  -- "b"."Pseudonym" as "Pseudonym",
  -- "b"."experience" as "experience"
FROM
  "Library"."Books" as "a"
  -- INNER JOIN "Library"."Author" as "b"
  -- ON "a"."Author" = "b"."id"
WHERE
  ("a"."id" < 77::bigint) AND
  ("a"."date_of_printing" >= '2001-04-09 00:00:00'::timestamp) AND
  ("a"."Name" LIKE 'NM'::varchar);
```

1 folder, 4 files: 64,1 KiB (65 632 bytes), Free space: 16,9 GiB

Результат:

```
id | date_of_printing | Name
68 | 2021-01-14 13:28:01.352239+02:00 | NM
72 | 2021-04-27 16:19:51.910497+03:00 | NM
2 rows, execution time: 0:00:00.000792
```

```
Books dynamic search interface
"id" < 77::bigint
"date_of_printing" >= '2001-04-09 00:00:00'::timestamp
"Name" LIKE 'NM'::varchar
```

## Код программного модуля model

AutoSchema.py

```
#!/usr/bin/env python

import re
import Lab.utils
import collections

# import collections
# import dataclasses
# import types
# import operator
import psycopg2
# import collections
# import pprint
# import re
# import itertools
# import more_itertools
# import numpy
# import click
# import pprint
import datetime
# import click_datetime
# import collections
import psycopg2.extensions
import psycopg2.sql

import Lab.utils.psql_types

__all__ = ["SchemaTable", "Schema"]

class SchemaTable(object):
    def __init__(self, schema=None, table=None):
        super().__init__()

        if table is None:
            table = type(self).__name__

        self.schema = schema
        self.table = table

        self.primary_key_name = f"id"

    def __str__(self):
        return f'"{self.table}"' if self.schema is None else
f'"{self.schema}"."{self.table}"'

    def __hash__(self):
        return hash(str(self))

    def columns(self):
        # sql = f"""
        #     SELECT column_name, data_type
        #     FROM information_schema.columns
        #     WHERE table_name = '{self.table}';
        # """

        sql = f"""
        SELECT
            tb.table_schema, tb.table_name, tb.column_name, tb.data_type,
```



```

tb.is_nullable,
        fx.constraint_name, fx.references_schema, fx.references_table,
fx.references_field
    FROM information_schema.columns tb
    LEFT JOIN (
        SELECT
            tc.constraint_schema,
            tc.table_name,
            kcu.column_name,
            tc.constraint_name,
            tc.constraint_type,
            rc.update_rule AS on_update,
            rc.delete_rule AS on_delete,
            ccu.constraint_schema AS references_schema,
            ccu.table_name AS references_table,
            ccu.column_name AS references_field
        FROM information_schema.table_constraints tc
        LEFT JOIN information_schema.key_column_usage kcu
            ON tc.constraint_catalog = kcu.constraint_catalog
            AND tc.constraint_schema = kcu.constraint_schema
            AND tc.constraint_name = kcu.constraint_name
        LEFT JOIN information_schema.referential_constraints rc
            ON tc.constraint_catalog = rc.constraint_catalog
            AND tc.constraint_schema = rc.constraint_schema
            AND tc.constraint_name = rc.constraint_name
        LEFT JOIN information_schema.constraint_column_usage ccu
            ON rc.unique_constraint_catalog = ccu.constraint_catalog
            AND rc.unique_constraint_schema = ccu.constraint_schema
            AND rc.unique_constraint_name = ccu.constraint_name
        WHERE tc.constraint_schema NOT ILIKE 'pg_%' AND tc.con-
straint_schema NOT ILIKE 'inform%' AND tc.constraint_type IN ('PRIMARY KEY',
'FOREIGN KEY')) fx
        ON fx.constraint_schema = tb.table_schema AND fx.table_name =
tb.table_name AND fx.column_name = tb.column_name
        WHERE tb.table_schema = '{self.schema}' AND tb.table_name =
'{self.table}'
    ORDER BY tb.ordinal_position;
"""
    # row_type(table_schema='Lab', table_name='Users', column_name='id',
data_type='bigint', is_nullable='NO', constraint_name='Users_pkey', refer-
ences_schema=None, references_table=None, references_field=None),
    with self.schema.dbconn.cursor() as dbcursor:
        dbcursor.execute(sql)
        row_type = collections.namedtuple("row_type", (a[0] for a in dbcur-
sor.description))
        result = tuple(row_type(*a) for a in dbcursor.fetchall())
        # result = {a: b for a, b in dbcursor.fetchall() if a not in
[f"{self.primary_key_name}"]}
        return result

    def describe(self):
        print(f"{self} describe")
        sql = f"""
        SELECT table_name, column_name, data_type, character_maximum_length
        FROM information_schema.columns
        WHERE table_schema = '{self.schema}' AND table_name = '{self.ta-
ble}';
        """
        return self.showData(sql=sql)

    def addData(self, data: dict[collections.namedtuple] = None):
        if data is None:
            return Lab.utils.menuInput(self.addData, [a for a in self.columns()
if a.column_name not in [f"{self.primary_key_name}"]])

```

```

        columns, values = zip(*{a.column_name: b for a, b in
data.items()}.items())

    sql = f"""
        INSERT INTO {self} (%s) VALUES %s;
    """

    with self.schema.dbconn.cursor() as dbcursor:
        try:
            dbcursor.execute(sql, (psycpg2.extensions.AsIs(",
".join(map(lambda x: f'"{x}"', columns))), values))
            self.schema.dbconn.commit()
        except Exception as e:
            self.schema.dbconn.rollback()
            print(f"Something went wrong: {e}")
            # raise e
        else:
            print(f"{dbcursor.rowcount} rows added")

    def editData(self, data: dict[collections.namedtuple] = None):
        if data is None:
            return Lab.utils.menuInput(self.editData, [a for a in self.columns()
if a.column_name not in []])

        tmp = next(a for a in data if a.column_name in [f"{self.pri-
mary_key_name}"])
        rowid = data[tmp]
        del data[tmp]

        columns, values = zip(*{a.column_name: b for a, b in
data.items()}.items())

        sql = f"""UPDATE {self} SET {", ".join(f'"{a}" = %s' for a in columns)}
WHERE "{self.primary_key_name}" = {rowid};"""

        with self.schema.dbconn.cursor() as dbcursor:
            try:
                dbcursor.execute(sql, values)
                self.schema.dbconn.commit()
            except Exception as e:
                self.schema.dbconn.rollback()
                print(f"Something went wrong: {e}")
            else:
                print(f"{dbcursor.rowcount} rows changed")

    def removeData(self, rowid=None):
        # rowid = click.prompt(f"{self.primary_key_name}", type=int)
        if rowid is None:
            return Lab.utils.menuInput(self.removeData, [a for a in self.col-
umns() if a.column_name in [f"{self.primary_key_name}"]])

        if isinstance(rowid, dict):
            rowid = rowid[next(a for a in rowid if a.column_name in
[f"{self.primary_key_name}"])]

        sql = f"""DELETE FROM {self} WHERE "{self.primary_key_name}" =
{rowid};"""

        with self.schema.dbconn.cursor() as dbcursor:
            try:
                dbcursor.execute(sql)
                self.schema.dbconn.commit()
            except Exception as e:

```

```

        self.schema.dbconn.rollback()
        print(f"Something went wrong: {e}")
    else:
        print(f"{dbcursor.rowcount} rows deleted")

def showData(self, sql=None):
    # print(showDataCreator)
    if sql is None:
        sql = f"""SELECT * FROM {self};"""

    return self.schema.showData(sql=sql)

def dynamicsearch(self):
    raise NotImplementedError

def randomFill(self, instances: int = None, str_len: int = 10, sql_re-
place: str = None):

    if sql_replace:
        pass
    else:
        if instances is None:
            return Lab.utils.menuInput(self.randomFill, [collections.namedtu-
ple("instances", ["column_name", "data_type", "default"])(instances, "int",
lambda: 100)])

        if isinstance(instances, dict):
            instances = instances[next(a for a in instances if a.column_name
in ["instances"])]

        columns = tuple(a for a in self.columns() if a.column_name not in
[f"{self.primary_key_name}"])

        def psql_foreign_key_random(x):
            result = f"""
                (SELECT "{x.references_field}" FROM "{x.refer-
ences_schema}"."{x.references_table}" ORDER BY random()*q LIMIT 1)
            """
            return result

        sql = ",\n"
        sql = f"""
INSERT INTO {self}({", ".join(map(lambda x: f'"{x.column_name}"',
columns))})
SELECT
    {sql.join(map(lambda x:
Lab.utils.psql_types.psql_types_to_random[x.data_type](x) if x.refer-
ences_field is None else psql_foreign_key_random(x), columns))}
FROM
    (VALUES ('qwertyuiopasdfghjklzxcvbnmQWERTYUI-
OPASDFGHJKLZXCVBNM')) as symbols(characters),
    generate_series(1, {instances}) as q;
"""

    sql = sql_replace or sql
    # with self.schema.dbconn:
    with self.schema.dbconn.cursor() as dbcursor:
        try:
            # print(sql)
            t1 = datetime.datetime.now()
            dbcursor.execute(sql)
            t2 = datetime.datetime.now()
            self.schema.dbconn.commit()
        except Exception as e:

```

```

        self.schema.dbconn.rollback()
        print(f"Something went wrong: {e}")
    else:
        print(f"{self} {dbcursor.rowcount} rows added, execution time:
{t2 - t1}")

    @property
    def prompt(self):
        return f"{self} table interface:"

    @property
    def __lab_console_interface__(self):
        result = Lab.utils.LabConsoleInterface({
            f"describe": self.describe,
            f"show data": self.showData,
            f"add data": self.addData,
            f"edit data": self.editData,
            f"remove data": self.removeData,
            f"random fill": self.randomFill,
            f"return": lambda: Lab.utils.menuReturn(f"User menu return"),
        }, prompt=self.prompt)
        return result

class SchemaTables(object):
    def __init__(self, schema, *tables):
        super().__init__()
        self.schema = schema
        self._tables = {str(a): (SchemaTable(self.schema, a) if isinstance(a,
str) else a) for a in tables}
        # self._iter = 0

    # @property
    # def tables(self):
    #     return self._tables.keys()

    def __str__(self):
        return f"{self.schema} ({type(self).__name__} ({set(self._ta-
bles.keys())}))"

    def __getattr__(self, name):
        try:
            if name in [f"_tables"]:
                raise KeyError
            return self._tables[name]
        except KeyError as e:
            try:
                return super().__getattr__(name)
            except KeyError as e:
                raise AttributeError(f"{name} is not known table")

    def __setattr__(self, key, value):
        if re.match(r"^[A-Z]$", key[0]):
            # print(f"sttr {key} {value}")
            self._tables[key] = value
        else:
            super().__setattr__(key, value)

    def __getitem__(self, key: str):
        try:
            return self._tables[key]
        except KeyError as e:
            raise KeyError(f"{key} is not known table")

```

```

def __setitem__(self, key, value):
    self._tables[key] = value

def __iter__(self):
    # self._iter = iter(self._tables.values())
    return iter(self._tables.values())

# def __next__(self):
#     try:
#         result = tuple(self._tables.values())[self._iter] # may be opti-
mized
#     except IndexError as e:
#         self._iter = 0
#         raise StopIteration
#     self._iter += 1
#     return result

# def __getitem__(self, key)

class Schema(object):
    def __init__(self, dbconn, name=None):
        super().__init__()
        if name is None:
            name = type(self).__name__
        self.dbconn = dbconn
        self.name: str = name
        self._tables: tuple = tuple()
        self._dynamicsearch: dict[str, DynamicSearchBase] = dict()
        self.refresh_tables()
        self.reoverride()

    def __str__(self):
        return self.name

    def __getitem__(self, key):
        return self.tables[key]

    def __iter__(self):
        return iter(self._tables)

    def showData(self, sql):
        with self.dbconn.cursor() as dbcursor:
            try:
                # print(sql)
                t1 = datetime.datetime.now()
                dbcursor.execute(sql)
                t2 = datetime.datetime.now()
            except Exception as e:
                self.dbconn.rollback()
                print(f"Something went wrong: {e}")
            else:
                q = Lab.utils.TablePrint()
                q.rowcount = dbcursor.rowcount
                q.table = Lab.utils.fetchall_table(dbcursor)
                q.executiontime = t2 - t1

                return q

    def reoverride(self):
        pass

    def refresh_tables(self):
        # self._tables: tuple = tuple()

```

```

sql = f"""
    SELECT table_name
    FROM information_schema.tables
    WHERE table_schema = '{self.name}';
"""

with self.dbconn.cursor() as dbcursor:
    dbcursor.execute(sql)
    q = (*a[0] for a in dbcursor.fetchall()),
    self._tables = SchemaTables(self, *q) # collections.namedtuple("Tables", q)(*map(SchemaTables, q))
    # pprint.pprint(self._tables)
    self.reoverride()
    return self._tables

def dump_sql(self):
    pass

def reinit(self):
    raise NotImplementedError(f"Need to override")

def randomFill(self):
    raise NotImplementedError(f"Need to override")

@property
def tables(self):
    return self._tables

@property
def dynamicsearch(self):
    return self._dynamicsearch

# def dynamicsearch(self):
#     raise NotImplementedError(f"Need to override")

@property
def prompt(self):
    return f'Schema "{self}" interface'

@property
def __lab_console_interface__(self):
    result = Lab.utils.LabConsoleInterface({
        **{f'{a.table}" table': (lambda a: lambda: a)(a) for a in self.tables},
        f'Schema "{self}" utils':
            lambda: Lab.utils.LabConsoleInterface({
                "reinit": self.reinit,
                "random fill": self.randomFill,
                # "dump sql": self.dump_sql,
                "return": lambda: Lab.utils.menuReturn(f"User menu return"),
            }, prompt=f'Schema "{self}" utils'),
        f"Dynamic search": lambda: Lab.utils.LabConsoleInterface({
            **{a: (lambda x: lambda: x)(b) for a, b in self.dynamicsearch.items()}},
            "return": lambda: Lab.utils.menuReturn(f"User menu return"),
            }, prompt=f'""Schema "{self}" dynamic search interface""')
            #self.dynamicsearch,
        }, prompt=self.prompt)

    return result

def _test():
    pass

```

```
if __name__ == "__main__":
    _test()
```

## DynamicSearch.py

```
#!/usr/bin/env python
import Lab.utils
import datetime
import itertools
import collections
import Lab.utils.psql_types

__all__ = [
    "CompareConstant",
    "SearchCriterias",
    "SelectCompositor",
    "DynamicSearchBase",
]

class CompareConstant(object):
    def __init__(self, psql_type, comparator=None, constant=None):
        super().__init__()
        self.comparator = comparator
        self._constant = None
        self._psql_type = psql_type

    def __str__(self):
        if self.isIgnored:
            return f"ignored"
        # if isinstance(self.constant, str) else self.constant
        return f"{self.comparator} {self.constant}::{self.psql_type}"

    def __repr__(self):
        return f"{type(self).__name__}(comparator={self.comparator}, constant={self.constant})"

    def reset(self):
        self.comparator = None
        self.setNull()

    def setNull(self):
        self.constant = None

    def setConstant(self, constant=None):
        if constant is None:
            return Lab.utils.menuInput(self.setConstant, [collections.namedtuple("instances", ["column_name", "data_type", "default"])(self.psql_type, self.psql_type, lambda: None)])
        else:
            self.constant = constant[next(a for a in constant if a.column_name in [self.psql_type])]
            # self.constant = click.prompt(self.psql_type,
            #                               type=Lab.utils.psql_types.psql_types_convert[self.psql_type].type,
            #                               default=Lab.utils.psql_types.psql_types_convert[self.psql_type].default(),
            #                               show_default=True)

    @property
    def isIgnored(self):
        return self.comparator is None

    @property
    def psql_type(self):
```

```

        return self._psql_type

@property
def constant(self):
    if isinstance(self._constant, (str, datetime.datetime)):
        return f"{{{self._constant}}}"
    elif self._constant is None:
        return f"NULL"
    # print(type(self._constant))
    return self._constant

@constant.setter
def constant(self, value):
    self._constant = value

def _lt(self):
    self.comparator = "<"

def _le(self):
    self.comparator = "<="

def _eq(self):
    self.comparator = "="

def _ne(self):
    self.comparator = "!="

def _ge(self):
    self.comparator = ">="

def _gt(self):
    self.comparator = ">"

def _like(self):
    self.comparator = "LIKE"

@property
def prompt(self) -> str:
    return f"Criteria editor: {self}"

@property
def __lab_console_interface__(self):
    result = Lab.utils.LabConsoleInterface({
        "ignore": self.reset,
        "<": self._lt,
        "<=": self._le,
        "=": self._eq,
        "!=": self._ne,
        ">=": self._ge,
        ">": self._gt,
        "LIKE": self._like,
        # "IS": lambda: setattr(self, "comparator", "IS"),
        # "IS NOT": lambda: setattr(self, "comparator", "IS NOT"),
        "set NULL": self.setNull,
        "set constant": self.setConstant,
        # "moar": lambda: self,
        "return": lambda: Lab.utils.menuReturn(f"User menu return"),
    }, prompt=self.prompt)
    return result

class SearchCriterias(list):
    def __init__(self, psql_mapping: str, psql_name: str, psql_type: str,
*args, **kwargs):

```



```

    super().__init__(*args, **kwargs)
    self._psql_mapping = psql_mapping
    self._psql_name = psql_name
    self._psql_type = psql_type

    @property
    def psql_mapping(self):
        return self._psql_mapping

    @property
    def psql_name(self):
        return self._psql_name

    @property
    def psql_type(self):
        return self._psql_type

    def reset(self):
        self.clear()

    def append(self):
        # if isinstance(obj, CompareConstant):
        #     return super().append(obj)
        # elif obj is None:
        #     return super().append(obj)
        # raise TypeError(f"{type(obj)} is invalid") CompareCon-
        stant(self.psql_type)
        # q = CompareConstant(self.psql_type)

        try:
            next(a for a, b in enumerate(self) if b.isIgnored)
        except StopIteration:
            super().append(CompareConstant(self.psql_type))

        return self

    def gen_sql(self):
        result = f""""{self.psql_mapping} {a}" for a in self if
not a.isIgnored)"""
        # print(f"{result=}")
        if result:
            result = f"({result})"
        return result

    @property
    def sql(self):
        return self.gen_sql()

    def __format__(self, format_=None):
        if format_ == "v":
            return f"{list(filter(lambda x: not (x.isIgnored), self))}"
        elif format_ == "sql":
            return self.gen_sql()
        elif format_ == "pre":
            result = f""""{self.psql_mapping} {a}" for a in self if not a.isI-
gnored)"""
            if result:
                return result
            return f"ignored"
        return super().__format__(format_)

    # def append_if_needed(self):
    # @property
    # def lab_console_interface(self):

```

```

# result = Lab.utils.LabConsoleInterface()
# result.update({f"Property {a} {b}": (lambda x: lambda: x)(b) for a, b
in enumerate(self, 1)})
# result.prompt = f"{self}"
# return result

class SelectCompositor(object):
    def __init__(self, search_criterias, table):
        super().__init__()
        self._search_criterias: SearchCriterias[CompareConstant] = search_cri-
terias
        self._table = table
        self.search_criterias.append()

    @property
    def table(self):
        return self._table

    @property
    def search_criterias(self):
        return self._search_criterias

    @property
    def prompt(self):
        return f'"{self.table}" {self.search_criterias:pre} select criterias:'

    @property
    def __lab_console_interface__(self):
        try:
            self.search_criterias.append()
            result = Lab.utils.LabConsoleInterface({
                **{f"Property {a} {b}": (lambda x: lambda: x)(b) for a, b in enu-
merate(self.search_criterias, 1)},
                # "new criteria": lambda: self.search_criterias[self.table].ap-
pend(),
                "return": lambda: Lab.utils.menuReturn(f"User menu return"),
            }, prompt=self.prompt)
            return result
        except Exception as e:
            print(e)

    def __bool__(self):
        return bool(self.search_criterias)

    # def reset(self):
    #     self.search_criterias.reset()

    # def __format__(self, *args, **kwargs):
    #     return self.search_criterias.__format__(*args, **kwargs)

class DynamicSearchBase(object):
    def __init__(self, schema):
        super().__init__()
        self.name = type(self).__name__
        self.schema = schema
        self._search: dict[SelectCompositor] = dict()
        # self.selectcompositors = tuple()

    @property
    def search(self) -> dict[SelectCompositor]:
        return self._search

```

```

    @search.setter
    def search(self, value: dict):
        self._search = dict(itertools.starmap(lambda key, value: (key, SelectCompositor(value, key)), value.items()))

    def execute(self) -> Lab.utils.TablePrint:
        return self.schema.showData(sql=self.sql)

    def reset(self) -> None:
        for a in self.search.values():
            a.search_criterias.reset()

    @property
    def where(self) -> str:
        newline = " AND \n"
        return newline.join(f"{a.search_criterias:sql}" for a in self.search.values() if f"{a.search_criterias:sql}")

    @property
    def sql(self) -> str:
        raise NotImplementedError(f"Need to override")

    @property
    def prompt(self):
        newline = f"\n"
        return f"""\{self.name} dynamic search interface\n{newline.join(f'"{a}"\{b.search_criterias:pre}' for a, b in self.search.items())}"""

    @property
    def __lab_console_interface__(self):
        try:
            result = Lab.utils.LabConsoleInterface({
                # **{f"{a}": (lambda x: lambda: SelectCompositor(self.search[x], x))(a) for a in self.search},
                **{a: (lambda x: lambda: x) (b) for a, b in self.search.items()},
                f"execute": self.execute,
                f"sql": lambda: print(self.sql),
                f"reset": self.reset,
                f"return": lambda: Lab.utils.menuReturn(f"User menu return"),
            }, prompt=self.prompt)
            return result
        except Exception as e:
            print(e)

def _test():
    pass

if __name__ == "__main__":
    _test()

```

#### Schema.py

```
#!/usr/bin/env python3
```

```
#!/usr/bin/env python3
```

```

from . import DynamicSearch
from .AutoSchema import *

```

```

class Library(Schema):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self._dynamicsearch = {a.name: a for a in [DynamicSearch.BooksDynamicSearch(self), DynamicSearch.ReaderLoanDynamicSearch(self), DynamicSearch.ReaderDynamicSearch(self), ]}
        # self.reoverride()

    def reoverride(self):
        pass

    def reinit(self):
        # sql = f"""
        #     SELECT table_name FROM information_schema.tables
        #     WHERE table_schema = '{self}';
        # """
        with self.dbconn.cursor() as dbcursor:
            # dbcursor.execute(sql)
            for a in self.refresh_tables(): # tuple(dbcursor.fetchall()):
                q = f"""DROP TABLE IF EXISTS {a} CASCADE;"""
                # print(q)
                dbcursor.execute(q)

tables = [
    f"""CREATE SCHEMA IF NOT EXISTS "{self}";""",
    f"""CREATE TABLE "{self}."Author" (
        "id" bigserial PRIMARY KEY,
        "Full_name" character varying(255) NOT NULL,
        "Pseudonym" character varying(255) NOT NULL,
        "experience" bigint NOT NULL
    );
    """,
    f"""CREATE TABLE "{self}."Reader" (
        "id" bigserial PRIMARY KEY,
        "Full Name" character varying(255) NOT NULL,
        "Age" bigint NOT NULL
    );
    """,
    f"""CREATE TABLE IF NOT EXISTS "{self}."Books" (
        "id" bigserial PRIMARY KEY,
        "date_of_printing" timestamp with time zone NOT NULL,
        "Name" character varying(255) NOT NULL,
        "Author" bigint NOT NULL,
        "Reader_id" bigint NOT NULL,
        CONSTRAINT "Books_Author_fkey" FOREIGN KEY ("Author")
            REFERENCES "{self}."Author"("id") MATCH SIMPLE
            ON UPDATE NO ACTION
            ON DELETE CASCADE
            NOT VALID,
        CONSTRAINT "Books_Reader_id_fkey" FOREIGN KEY ("Reader_id")
            REFERENCES "{self}."Reader"("id") MATCH SIMPLE
            ON UPDATE NO ACTION
            ON DELETE CASCADE
            NOT VALID
    );
    """,
    f"""CREATE TABLE IF NOT EXISTS "{self}."Library" (
        "id" bigserial PRIMARY KEY,
        "year_of_foundation" timestamp with time zone NOT NULL,
        "address" character varying(255) NOT NULL,
        "capacity" bigint NOT NULL,
        "Reader_id" bigint NOT NULL,

```

```

        CONSTRAINT "Library_Reader_id_fkey" FOREIGN KEY ("Reader_id")
            REFERENCES "{self}."Reader("id") MATCH SIMPLE
            ON UPDATE NO ACTION
            ON DELETE CASCADE
            NOT VALID
    );
    """
    f"""CREATE TABLE IF NOT EXISTS "{self}."LibraryBooks" (
        "id" bigserial PRIMARY KEY,
        "Library_id" bigint NOT NULL,
        "Book_id" bigint NOT NULL,
        CONSTRAINT "LibraryBooks_Library_id_fkey" FOREIGN KEY ("Li-
brary_id")
            REFERENCES "{self}."Library("id") MATCH SIMPLE
            ON UPDATE NO ACTION
            ON DELETE CASCADE
            NOT VALID,
        CONSTRAINT "LibraryBooks_Book_id_fkey" FOREIGN KEY ("Book_id")
            REFERENCES "{self}."Books("id") MATCH SIMPLE
            ON UPDATE NO ACTION
            ON DELETE CASCADE
            NOT VALID
    );
    """
]

with self.dbconn.cursor() as dbcursor:
    for a in tables:
        # print(a)
        dbcursor.execute(a)

self.dbconn.commit()

tables = self.refresh_tables()
# print(f"tables: {tables}")

def randomFill(self):
    self.tables.Author.randomFill(1_000)
    self.tables.Reader.randomFill(1_000)
    self.tables.Books.randomFill(1_000)
    self.tables.Library.randomFill(1_000)
    self.tables.LibraryBooks.randomFill(1_000)

def _test():
    pass

if __name__ == "__main__":
    _test()

```

### \_\_init\_\_.py

```

#!/usr/bin/env python
from .labmenu import *

def _test() -> None:
    pass

if __name__ == "__main__":
    _test()


```

### **Короткий опис функцій**

Класи таблиць створюються автоматично, виходячи з інформації з бази даних. Класи таблиць відповідно мають в своєму складі функції для роботи з відповідними таблицями у базі даних, кожен з класів має такі функції з запитами до бази даних:

1. `addData` – додає рядок даних до таблиці
2. `editData` – дозволяє змінити рядок даних в таблиці
3. `removeData` – видаляє рядок з таблиці
4. `showData` – виводить таблицю
5. `randomFill` – генерація випадкових даних у таблицю

## Ілюстрації програмного коду на Github

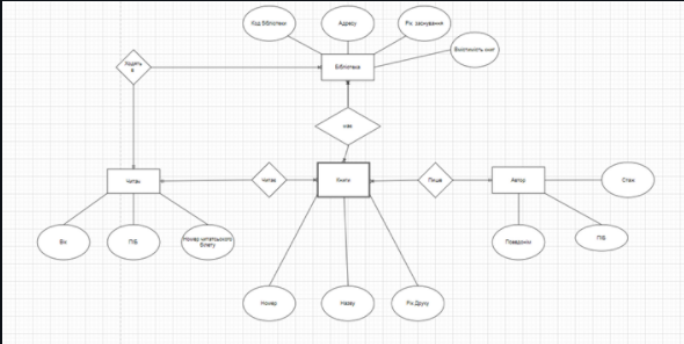
 DmytroZhyla Delete Дмитро\_Жила\_KB\_94\_Лабораторна\_робота\_№2.pdf 342bfeb 28 seconds ago 5 commits

Lab	Add files via upload	7 minutes ago
.gitignore	Add files via upload	9 minutes ago
README.md	Update README.md	15 minutes ago
lab2.py3	Add files via upload	9 minutes ago
postgres.ini.secure	Add files via upload	9 minutes ago

README.md

### BD\_LAB2

Лабораторна робота №2 з дисципліни «Бази даних і засоби управління» Тема: «Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL» Студент групи KB-94 Жила Дмитро



Лабораторна робота №2 з дисципліни «Бази даних і засоби управління» Тема: «Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL» Студент групи KB-94 Жила Дмитро

Readme

0 stars

1 watching

0 forks

#### Releases

No releases published

[Create a new release](#)

#### Packages

No packages published

[Publish your first package](#)

#### Languages

Python 100.0%

Посилання на репозиторій: [https://github.com/DmytroZhyla/BD\\_LAB2](https://github.com/DmytroZhyla/BD_LAB2)