



НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КІЇВСЬКИЙ 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-94

Орел Б.В.

Перевірив: доц. Петрашенко А. В.

Київ – 2021

Постановка задачі

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

Загальне завдання роботи полягає у наступному:

1. Реалізувати функції перегляду, 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 атрибутиами з двох та більше сущностей 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).

Інформація про програму

Посилання на репозиторій у GitHub з вихідним кодом програми та звітом:

https://github.com/OrelBogdan/lab2_bd

Використана мова програмування: Python 3.9

Використані бібліотеки: psycopg2 (для зв'язку з СУБД), time (для виміру часу запиту пошуку для завдання 3), sys (для реалізації консольного інтерфейсу).

Відомості про обрану предметну галузь з лабораторної роботи №1

В концептуальній моделі предметної області “Магазин”(Рисунок 1) виділяються наступні сутності та зв'язки між ними:

1. Сутність “Categories” з атрибутами: Category_id, Category_name, власник, країна. Призначена для зберігання інформації про категорії магазину.
2. Сутність “Products” з атрибутами: Category_id , Manufacturer_id, Product_id, Product_name, Price, Amount. Зберігання інформації про товари кожної з категорій.
3. Сутність “Manufacturer” з атрибутами: Manufacturer_id, Manufacturer_name. Зберігає інформації про виробників товарів.
4. Сутність “Ordered_product” з атрибутами: Ordered_product_id, Product_id, Ordered_amount, Order_id. Зберігає інформацію про товари що були замовлені.
5. Сутність “Order” з атрибутами: Order_id, User_data_id. Зберігає інформацію про замовлення клієнта (один клієнт може зробити декілька замовлень, а кожне замовлення може складатися з декількох товарів)
6. Сутність “User” з атрибутами: User_data_id, Name, Surname, Patronymic, Email. Ця сутність призначена для збереження даних про замовника.

Одна категорія складається з багатьох товарів, тому між сутностями “Categories ” та “Products ” зв’язок R(1:N).

Один виробник, може створювати багато товарів , тому між сутностями “Manufacturer ” та “Products ” зв’язок R(1:N).

Кожен клієнт може зробити декілька замовлень тому між сутностями “User” та “Order” зв’язок R(1:N).

Так як одне замовлення може складатися з декількох товарів і один товар може бути замовлений одночасно у декількох замовленнях тому між сутностями “Products” та “Order” зв’язок R(N:M). Для його реалізації між сутностями “Products” та “Order” використовується сутність “Ordered_product”. Сутності “Products” та “Ordered_product” мають зв’язок R(1:N). Сутності “Order” та “Ordered_product” мають зв’язок R(1:N).

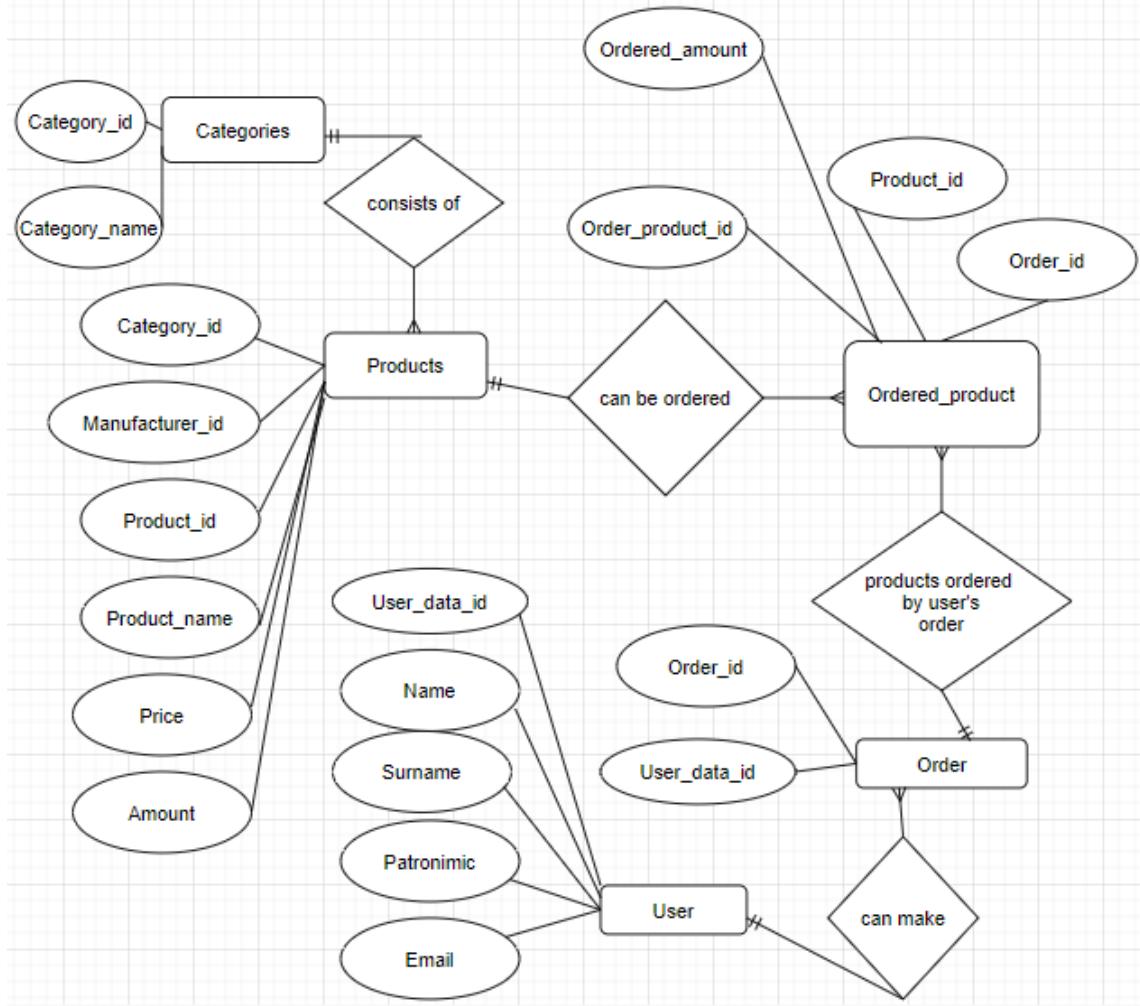


Рисунок 1 - Концептуальна модель предметної області “Магазин”.

Логічна модель (схема) БД “Магазин”

В логічній моделі (Рисунок 2):

1. Сутність “Categories” перетворена в таблицю “Categories”;
2. Сутність “Products” перетворена в таблицю “Products”;
3. Сутність “Manufacturer” перетворена в таблицю “Manufacturer”;
4. Сутність “Ordered_product” перетворена в таблицю “Ordered_product”;
5. Сутність “Order” перетворена в таблицю “Order”;
6. Сутність “User” перетворена в таблицю “User”.

Зв’язок R(N:M) між сутностями “Order” та “Products” зумовив появу таблиці “Ordered_product”

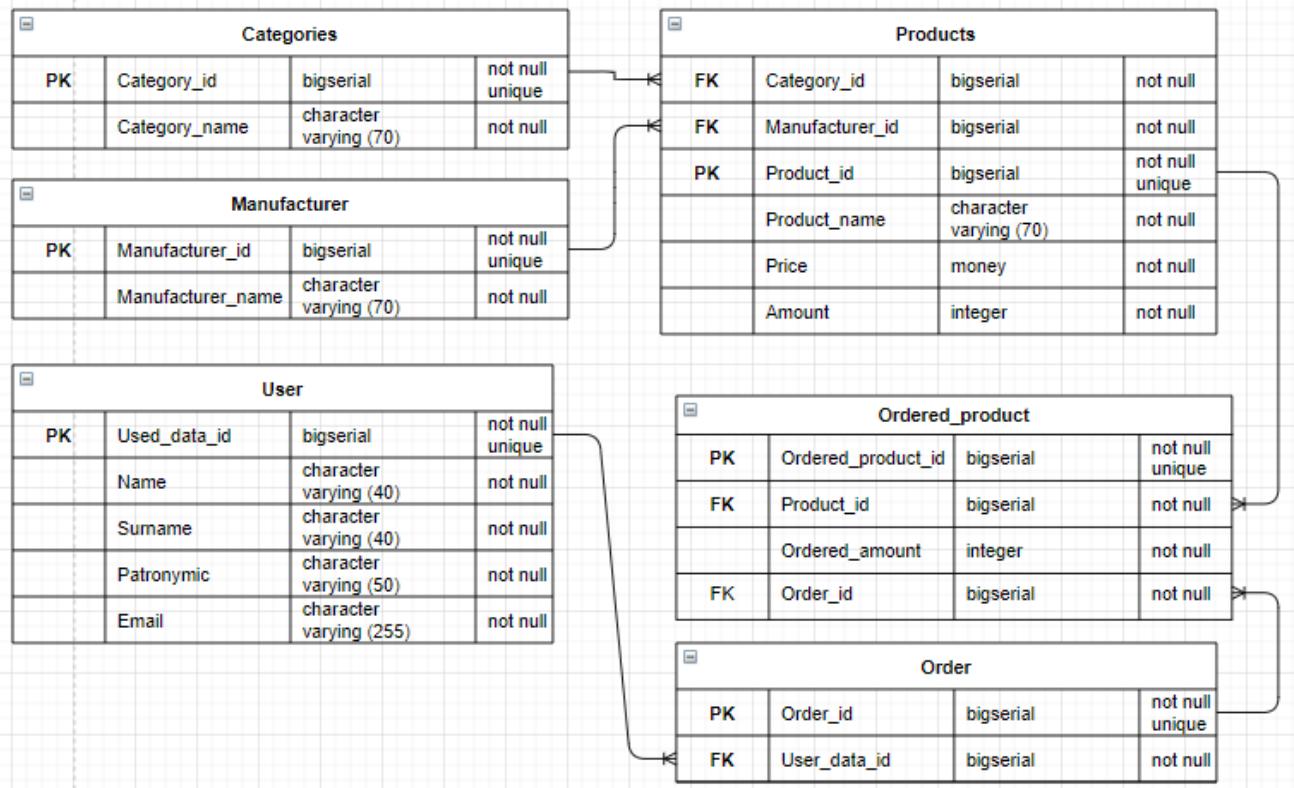


Рисунок 2 - Логічна модель предметної області “Магазин ”.

Опис структури БД

<i>Сутність</i>	<i>Атрибут</i>	<i>Опис атрибуту</i>	<i>Type</i>	<i>Обмеження</i>
Categories (інформація про категорії товарів)	Category_id	унікальний ID категорії	bigserial	not null unique
	Category_name	назва категорії	character varying(70)	not null
Products (інформація про товари)	Category_id	ID категорії	bigserial	not null unique
	Manufacturer_id	ID виробника	bigserial	not null
	Product_id	унікальний ID товару	bigserial	not null
	Product_name	назва товару	character varying(70)	not null
	Price	ціна	money	not null
	Amount	кількість на складі	integer	not null
Manufacturer (інформація про виробника товарів)	Manufacturer_id	унікальний ID виробника	bigserial	not null unique
	Manufacturer_name	ім'я виробника	character varying(70)	not null
Ordered_product (інформація про замовлені товари)	Ordered_product_id	унікальний ID замовленого товару	bigserial	not null unique
	Product_id	ID товару	bigserial	not null
	Ordered_amount	Кількість товару у замовленні	integer	not null
	Order_id	ID замовлення	bigserial	not null
Order (інформація про замовлення клієнтів)	Order_id	унікальний ID замовлення	bigserial	not null unique
	User_data_id	ID клієнта	bigserial	not null
User (інформація про клієнтів)	Used_data_id	унікальний ID клієнта	bigserial	not null unique
	Name	ім'я клієнта	character varying(40)	not null
	Surname	прізвище	character varying(40)	not null
	Patronymic	побатькові	character varying(50)	not null
	Email	електронна пошта	character varying(255)	not null

Схема меню користувача

```
[bohdan@bohdan Databases-and-Management-Tools-Lab2]$ python lab2.py3
MVC schema "Shop" interface
> "Categories" table
  "Products" table
  "Manufacturer" table
  "Order" table
  "Ordered_product" table
  "User" table
Schema "Shop" utils
Dynamic search
exit
```

На знімку екрану терміналу продемонстровано основне меню яке відображається при запуску програми, у кожному пункті основного меню знаходитьться підменю у пунктах “Categories” table, “Products” table, “Manufacturer” table, “Order” table, “Ordered_product” table, “User” table, пункти підменю однакові, а same describe, show data, add data, edit data, remove data, random fill, return.

Наприклад підменю у пункті “Categories” table

```
[bohdan@bohdan Databases-and-Management-Tools-Lab2]$ python lab2.py3
"Shop"."Categories" table interface:
> describe
  show data
  add data
  edit data
  remove data
  random fill
  return
```

describe – описує структуру таблиці типи даних назви колонок таблиці

show data – відображає дані таблиці

edit data – редагування таблиці

add data – додати данні у таблицю

remove data – видалити данні з таблиці

random fill – заповнити рандомізованими даними таблицю

return – вийти у початкове меню

Підменю у пункті Schema “Shop” utils

```
[bohdan@bohdan Databases-and-Management-Tools-Lab2]$ python lab2.py3
Schema "Shop" utils
> reinit
  random fill
  return
```

reinit – переініціалізовує базу (створює нову базу даних попередньо видаливши стару)

random fill – заповнює рандомізованими даними всі таблиці бази даних

return – вийти у початкове меню

Підменю у пункті Dynamic search

```
[bohdan@bohdan Databases-and-Management-Tools-Lab2]$ python lab2.py3
Schema "Shop" dynamic search interface
> CategoriesProducts
ManufacturerProducts
UserOrderedProducts
return
```

CategoriesProducts – пошук у таблицях “Shop”. “Categories” та “Shop”.“Products” одночасно за критеріями з двох таблиць

ManufacturerProducts – пошук у таблицях “Shop”. “Manufacturer” та “Shop”.“Products” одночасно за критеріями з двох таблиць

UserOrderedProducts – пошук у таблицях “Shop”. “User” та “Shop”.“Order”, “Shop”.“Ordered_product”,“Shop”.“Products”

return – вийти у початкове меню

що показує усі доступні користувачу команди, коротко описує їх та надає список

обов’язкових аргументів. Кожна команда запускає відповідний метод об’єкту класу Controller, який реалізує передачу аргументів у клас View на перевірку і за

умови їх коректності, Controller далі передає ці аргументи у клас Model, що здійснює запит до бази даних.

Завдання 1

Запит на видалення

Для перевірки роботи розглянемо запити на видалення дочірньої таблиці “Products” та батьківської таблиці “Categories”.

Таблиця “Products” до та після видалення запису 5:

```
[bohdan@bohdan Databases-and-Management-Tools-Lab2]$ python lab2.py3
SELECT * FROM "Shop"."Products";
+-----+-----+-----+-----+-----+-----+
| Product_id | Category_id | Manufacturer_id | Product_name | Price | Amount |
+-----+-----+-----+-----+-----+-----+
| 2           | 3           | 3               | zxcvbnmQWE   | 14.00 | 57      |
| 3           | 4           | 3               | rtyuiopasd  | 85.00 | 26      |
| 4           | 1           | 2               | SDFGHJKLZX  | 83.00 | 13      |
| 5           | 4           | 2               | BNM          | 34.00 | 60      |
+-----+-----+-----+-----+-----+
4 rows, execution time: 0:00:00.000919
Product_id: 5
DELETE FROM "Shop"."Products" WHERE "Product_id" = 5;
1 rows deleted
SELECT * FROM "Shop"."Products";
+-----+-----+-----+-----+-----+-----+
| Product_id | Category_id | Manufacturer_id | Product_name | Price | Amount |
+-----+-----+-----+-----+-----+-----+
| 2           | 3           | 3               | zxcvbnmQWE   | 14.00 | 57      |
| 3           | 4           | 3               | rtyuiopasd  | 85.00 | 26      |
| 4           | 1           | 2               | SDFGHJKLZX  | 83.00 | 13      |
+-----+-----+-----+-----+-----+
3 rows, execution time: 0:00:00.000492
```

Таблиця “Categories” спроба видалення запису 1:

```
+ rows deleted
SELECT * FROM "Shop"."Products";

Product_id | Category_id | Manufacturer_id | Product_name | Price   | Amount
2          | 3           | 3               | zxcvbnmQWE  | 14.00 ₴ | 57
3          | 4           | 3               | rtyuiopasd | 85.00 ₴ | 26
4          | 1           | 2               | SDFGHJKLZX | 83.00 ₴ | 13
3 rows, execution time: 0:00:00.000492

Category_id | Category_name
1           | jklzxcvbnm
2           | lzxcvbnmQW
3           | pasdfghjkl
4           | ghjklzxcvb
4 rows, execution time: 0:00:00.000688
Category_id: 1
DELETE FROM "Shop"."Categories" WHERE "Category_id" = 1;
Something went wrong: ОШИБКА: UPDATE или DELETE в таблице "Categories" нарушает ограничение внешнего ключа "Products_Category_id_fkey" таблицы "Products"
DETAIL: На ключ (Category_id)=(1) всё ещё есть ссылки в таблице "Products".
```

Видалення запису з батьківської таблиці, який зв’язаний з дочірньою таблицею, не буде здійснено, а буде видано повідомлення про помилку.

Запит на вставку поля

Для перевірки роботи розглянемо запити на вставки в дочірню таблицю “Products”. Спочатку коректний, потім з неіснуючим значенням зовнішнього ключа батьківської таблиці “Categories”.

Таблиця “Products”. до вставки запису

```
SELECT * FROM "Shop"."Products";

Product_id | Category_id | Manufacturer_id | Product_name | Price   | Amount
2          | 3           | 3               | zxcvbnmQWE  | 14.00 ₴ | 57
3          | 4           | 3               | rtyuiopasd | 85.00 ₴ | 26
4          | 1           | 2               | SDFGHJKLZX | 83.00 ₴ | 13
6          | 4           | 3               | Name        | 40.00 ₴ | 50
7          | 1           | 1               | rtyu        | 56.00 ₴ | 67
8          | 1           | 1               | Name        | 3.00 ₴  | 3
9          | 1           | 1               | wer         | 4.00 ₴  | 4
7 rows, execution time: 0:00:00.000532
```

Таблиця “Products” після вставки запису

```
Manufacturer_id: 1
Category_id: 6
Product_name: Name200
Price: 45
Amount: 300

INSERT INTO "Shop"."Products" ("Category_id", "Manufacturer_id",
"Product_name", "Price", "Amount") VALUES (6, 1,'Name200', 45.0, 300);

1 rows added
SELECT * FROM "Shop"."Products";

Product_id | Category_id | Manufacturer_id | Product_name | Price   | Amount
3          | 4           | 3               | rtyuiopasd | 85.00 ₴ | 26
4          | 1           | 2               | SDFGHJKLZX | 83.00 ₴ | 13
6          | 4           | 3               | Name        | 40.00 ₴ | 50
7          | 1           | 1               | rtyu        | 56.00 ₴ | 67
8          | 1           | 1               | Name        | 3.00 ₴  | 3
9          | 1           | 1               | wer         | 4.00 ₴  | 4
2          | 1           | 1               | Name3       | 3.00 ₴  | 3
10         | 6           | 1               | Name200     | 45.00 ₴ | 300
8 rows, execution time: 0:00:00.000509
```

Батьківська таблиця “Categories”

```
SELECT * FROM "Shop"."Categories";  
  
Category_id | Category_name  
1           | jklzxcvbnm  
2           | lzxcvbnmQW  
3           | pasdfghjkl  
4           | ghjklzxcvb  
5           | Name1  
6           | Name456  
6 rows, execution time: 0:00:00.000358
```

Спроба вставки запису у дочірню таблицю “Products” з неіснуючим зовнішнім ключем

```
Manufacturer_id: 1  
Category_id: 7  
Product_name: name300  
Price: 345  
Amount: 4  
  
INSERT INTO "Shop"."Products" ("Category_id", "Manufacturer_id", "Product_name", "Price", "Amount") VALUES (7, 1, 'name300', 345.0, 4);  
  
Something went wrong: ОШИБКА: INSERT или UPDATE в таблице "Products" нарушает ограничение внешнего ключа "Products_Category_id_fkey"  
DETAIL: Ключ (Category_id)=(7) отсутствует в таблице "Categories".
```

Запит на зміну поля

Для перевірки роботи розглянемо запити на зміну значення в дочірній таблиці “Ordered_product”. Спочатку коректний, потім з неіснуючим значенням зовнішнього ключа батьківської таблиці “Order”.

Записи у батьківській таблиці “Order” та таблиця “Ordered_product” до і після зміни

```
SELECT * FROM "Shop"."Ordered_product";  
  
Ordered_product_id | Product_id | Ordered_amount | Order_id  
1                 | 2          | 5             | 1  
2                 | 3          | 9             | 2  
2 rows, execution time: 0:00:00.000708  
SELECT * FROM "Shop"."Order";  
  
Order_id | User_data_id  
1        | 1  
2        | 2  
3        | 3  
3 rows, execution time: 0:00:00.000319  
Ordered_product_id: 1  
Product_id: 3  
Ordered_amount: 10  
Order_id: 3  
UPDATE "Shop"."Ordered_product" SET "Product_id" = 3, "Ordered_amount" = 10,  
"Order_id" = 3 WHERE "Ordered_product_id" = 1;  
1 rows changed  
SELECT * FROM "Shop"."Ordered_product";  
  
Ordered_product_id | Product_id | Ordered_amount | Order_id  
2                 | 3          | 9             | 2  
1                 | 3          | 10            | 3  
2 rows, execution time: 0:00:00.000671
```

Записи у батьківській таблиці “Order” та зміна за неіснуючим ключем таблиці “Ordered_product”

```
SELECT * FROM "Shop"."Order";  
  
Order_id | User_data_id  
1        | 1  
2        | 2  
3        | 3  
3 rows, execution time: 0:00:00.000566  
SELECT * FROM "Shop"."Ordered_product";  
  
Ordered_product_id | Product_id | Ordered_amount | Order_id  
2                 | 3          | 9             | 2  
1                 | 3          | 10            | 3  
2 rows, execution time: 0:00:00.000452  
Ordered_product_id: 1  
Product_id: 3  
Ordered_amount: 11  
Order_id: 4  
UPDATE "Shop"."Ordered_product" SET "Product_id" = 3, "Ordered_amount" = 11,  
"Order_id" = 4 WHERE "Ordered_product_id" = 1;  
Something went wrong: ОШИБКА: INSERT или UPDATE в таблице "Ordered_product"  
нарушает ограничение внешнего ключа "Ordered_product_Order_id_fkey"  
DETAIL: Ключ (Order_id)=(4) отсутствует в таблице "Order".
```

Завдання 2

Вставка 5 псевдорандомізованих записів у кожну з таблиць.

Початкова таблиця “Categories”

```
SELECT * FROM "Shop"."Categories";
```

Category_id	Category_name
1	jklzxcvbnm
2	lzxcvbnmQW
3	pasdfghjkl
4	ghjklzxcvb
5	Name1
6	Name456

Модифікована таблиця “Categories”

```
SELECT * FROM "Shop"."Categories";
```

Category_id	Category_name
1	jklzxcvbnm
2	lzxcvbnmQW
3	pasdfghjkl
4	ghjklzxcvb
5	Name1
6	Name456
7	d f g h j k l z x c
8	g h j k l z x c v b
9	h j k l z x c v b n
10	z x c v b n m Q W E
11	x c v b n m Q W E R

Запит

```
INSERT INTO "Shop"."Categories"("Category_name")
    SELECT
        substr(characters, (random() * length(characters) +
1)::integer, 10)
    FROM
        (VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters),
        generate_series(1, 5);
```

Початкова таблиця “Manufacturer”

```
SELECT * FROM "Shop"."Manufacturer";
```

Manufacturer_id	Manufacturer_name
1	LZXCVBNM
2	IOPASDFGHJ
3	KLZXCVBNM

Модифікована таблиця “Manufacturer”

SELECT * FROM "Shop"."Manufacturer";	
Manufacturer_id	Manufacturer_name
1	LZXCVBNM
2	IOPASDFGHJ
3	KLZXCVBNM
4	CVBNM
5	klzxcvbnmQ
6	zxcvbnmQWE
7	pasdfghjkl
8	VBNM

Запит

```
INSERT INTO "Shop"."Manufacturer"("Manufacturer_name")
    SELECT
        substr(characters, (random() * length(characters) +
1)::integer, 10)
    FROM
        (VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCV
BNM')) as symbols(characters),
        generate_series(1, 5);
```

Початкова таблиця “Ordered_product”

SELECT * FROM "Shop"."Ordered_product";			
Ordered_product_id	Product_id	Ordered_amount	Order_id
2	3	9	2
1	3	10	3

Модифікована таблиця “Ordered_product”

SELECT * FROM "Shop"."Ordered_product";			
Ordered_product_id	Product_id	Ordered_amount	Order_id
2	3	9	2
1	3	10	3
4	3	35	2
5	9	12	3
6	10	40	2
7	7	73	2
8	8	39	1

Запит

```

INSERT INTO "Shop"."Ordered_product"("Product_id", "Ordered_amount",
"Order_id")
    SELECT
        (SELECT "Product_id" FROM "Shop"."Products"
ORDER BY random()*q LIMIT 1),
        trunc(random() * 100)::int,
        (SELECT "Order_id" FROM "Shop"."Order"
ORDER BY random()*q LIMIT 1)

    FROM
        generate_series(1,5 ) as q;

```

Початкова таблиця “Order”

SELECT * FROM "Shop"."Order";	
Order_id	User_data_id
1	1
2	2
3	3

Модифікована таблиця “Order”

SELECT * FROM "Shop"."Order";	
Order_id	User_data_id
1	1
2	2
3	3
4	3
5	3
6	3
7	3
8	3

Запит

```

INSERT INTO "Shop"."Order"("User_data_id")
    SELECT
        (SELECT "User_data_id" FROM "Shop"."User"
ORDER BY random()*q LIMIT 1)

    FROM
        generate_series(1, 5) as q;

```

Початкова таблиця “User”

```
SELECT * FROM "Shop"."User";
```

User_data_id	Name	Surname	Patronymic	Email
1	User1	Surname1	Patronymic1	Email1
2	User2	Surname2	Patronymic2	Email2
3	User3	Surname3	Patronymic3	Email3

Модифікована таблиця “User”

```
SELECT * FROM "Shop"."User";
```

User_data_id	Name	Surname	Patronymic	Email
1	User1	Surname1	Patronymic1	Email1
2	User2	Surname2	Patronymic2	Email2
3	User3	Surname3	Patronymic3	Email3
4	yuiopasdfg	xcvbnmQWER	ERTYUIOPAS	VBNM
5	opasdfghjk	dfghjklzxc	KLZXCVBNM	nmQWERTYUI
6	hjklzxcvbn	hjklzxcvbn	zxcvbnmQWE	VBNM
7	QWERTYUIOP	yuiopasdfg	LZXCVBNM	xcvbnmQWER
8	tyuiopasdf	SDFGHJKLZX	ASDFGHJKLZ	mQWERTYUIO

Запит

```
INSERT INTO "Shop"."User"("Name", "Surname", "Patronymic", "Email")
    SELECT
        substr(characters, (random() * length(characters) +
1)::integer, 10),
        substr(characters, (random() * length(characters) +
1)::integer, 10),
        substr(characters, (random() * length(characters) +
1)::integer, 10),
        substr(characters, (random() * length(characters) +
1)::integer, 10)
    FROM
        (VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCV
BNM')) as symbols(characters);
```

Початкова таблиця “Products”

```
SELECT * FROM "Shop"."Products";
```

Product_id	Category_id	Manufacturer_id	Product_name	Price	Amount
3	4	3	rtyuiopasd	85.00 ₴	26
4	1	2	SDFGHJKLZX	83.00 ₴	13
6	4	3	Name	40.00 ₴	50
7	1	1	rtyu	56.00 ₴	67
8	1	1	Name	3.00 ₴	3
9	1	1	wer	4.00 ₴	4
2	1	1	Name3	3.00 ₴	3
10	6	1	Name200	45.00 ₴	300

Модифікована таблиця “Products”

```
SELECT * FROM "Shop"."Products";
```

Product_id	Category_id	Manufacturer_id	Product_name	Price	Amount
3	4	3	rtyuiopasd	85.00 ₴	26
4	1	2	SDFGHJKLZX	83.00 ₴	13
6	4	3	Name	40.00 ₴	50
7	1	1	rtyu	56.00 ₴	67
8	1	1	Name	3.00 ₴	3
9	1	1	wer	4.00 ₴	4
2	1	1	Name3	3.00 ₴	3
10	6	1	Name200	45.00 ₴	300
12	3	8	HJKLZXCVBN	43.00 ₴	24
13	16	4	uiopasdfgh	38.00 ₴	65
14	1	1	TYUIOPASDF	23.00 ₴	3
15	6	7	yuiopasdfg	37.00 ₴	70
16	14	2	tyuiopasdf	82.00 ₴	35

Запит

```
INSERT INTO "Shop"."Products"("Category_id", "Manufacturer_id",
"Product_name", "Price", "Amount")
SELECT
    (SELECT "Category_id" FROM "Shop"."Categories"
ORDER BY random()*q LIMIT 1),
    (SELECT "Manufacturer_id" FROM
"Shop"."Manufacturer" ORDER BY random()*q LIMIT 1),
    substr(characters, (random() * length(characters) +
1)::integer, 10),
    trunc(random() * 100)::int,
    trunc(random() * 100)::int
FROM
    (VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCV
BNM')) as symbols(characters),
    generate_series(1, 5) as q;
```

Завдання 3

Пошук за атрибутами “Price”,“Amount”,“Category_name” з двох таблиць (“Products”, “Categories”).

Формування запиту:

```
money: 50
money: 40
integer: 80
varchar: Name456
CategoriesProducts dynamic search interface
"Category_name" LIKE 'Name456'::varchar
"Price" <= 50.0::money AND >= 40.0::money
"Amount" >= 80::integer
    Category_name
    Price
> Amount
execute
sql
reset
return
```

Результат:

Category_name	Category_id	Product_id	Product_name	Price	Amount
Name456	6	10	Name200	45.00 ₴	300

1 rows, execution time: 0:00:00.009375

Запит:

SELECT

```

    "a"."Category_name",
    "a"."Category_id",
    "b"."Product_id",
    "b"."Product_name",
    "b"."Price",
    "b"."Amount"

FROM
    "Shop"."Categories" as "a"
INNER JOIN "Shop"."Products" as "b"
    ON "a"."Category_id" = "b"."Category_id"

WHERE
    ("a"."Category_name" LIKE 'Name456'::varchar) AND
    ("b"."Price" <= 50.0::money AND "b"."Price" >= 40.0::money) AND
    ("b"."Amount" >= 80::integer);
```

Пошук за атрибутами “Manufacturer_name”, “Price”, “Product_name” з двох таблиць (“Manufacturer”, “Categories”).

Формування запиту:

```
varchar: Name3
money: 3
money: 80
varchar: LZXCVBNM
ManufacturerProducts dynamic search interface
"Manufacturer_name" LIKE 'LZXCVBNM'::varchar
"Price" >= 3.0::money AND <= 80.0::money
"Product_name" >= 'Name3'::varchar
    Manufacturer_name
    Price
> Product_name
execute
sql
reset
return
```

Результат:

Manufacturer_name	Manufacturer_id	Product_id	Product_name	Price	Amount
LZXCVBNM	1	7	rtyu	56.00 ₴	67
LZXCVBNM	1	9	wer	4.00 ₴	4
LZXCVBNM	1	2	Name3	3.00 ₴	3
LZXCVBNM	1	14	TYUIOPASDF	23.00 ₴	3

Запит:

SELECT

```
"a"."Manufacturer_name",
"a"."Manufacturer_id",
"b"."Product_id",
"b"."Product_name",
"b"."Price",
"b"."Amount"
```

FROM

```
"Shop"."Manufacturer" as "a"
INNER JOIN "Shop"."Products" as "b"
    ON "a"."Manufacturer_id" = "b"."Manufacturer_id"
```

WHERE

```
("a"."Manufacturer_name" LIKE 'LZXCVBNM'::varchar)
```

AND

```
("b"."Price" >= 3.0::money AND "b"."Price" <= 80.0::money) AND
("b"."Product_name" >= 'Name3'::varchar);
```

Пошук за атрибутами “ Name”,“ Surname”,“ Ordered_amount” з чотирьох таблиць (“User”, “Order”, “Ordered_product”,“Products”).

Формування запиту:

```
1 varchar: User1
2 varchar: Surname3
3 integer: 70
4 UserOrderedProducts dynamic search interface
5 "Name" >= 'User1'::varchar
6 "Surname" <= 'Surname3'::varchar
7 "Ordered_amount" <= 70::integer
8   Name
9   Surname
> Ordered_amount
10  execute
11  sql
12  reset
13  return
```

Результат:

Ordered_product_id	Ordered_amount	Order_id	Name	Surname	Patronymic	Product_name
2	9	2	User2	Surname2	Patronymic2	rtyuiopasd
1	10	3	User3	Surname3	Patronymic3	rtyuiopasd
5	12	3	User3	Surname3	Patronymic3	wer
6	40	2	User2	Surname2	Patronymic2	Name200
8	39	1	User1	Surname1	Patronymic1	Name

5 rows, execution time: 0:00:00.001161

Запит:

```
SELECT
    "a"."Ordered_product_id",
    "a"."Ordered_amount",
    "a"."Order_id",
    "c"."Name",
    "c"."Surname",
    "c"."Patronymic",
    "d"."Product_name"
FROM
    "Shop"."Ordered_product" as "a"
INNER JOIN "Shop"."Order" as "b"
    ON "a"."Order_id" = "b"."Order_id"
```

```
INNER JOIN "Shop"."User" as "c"
    ON "b"."User_data_id" = "c"."User_data_id"
INNER JOIN "Shop"."Products" as "d"
    ON "a"."Product_id" = "d"."Product_id"
WHERE
    ("c"."Name" >= 'User1'::varchar) AND
    ("c"."Surname" <= 'Surname3'::varchar) AND
    ("a"."Ordered_amount" <= 70::integer);
```

Завдання 4

Код програмного модулю model
Schema.py

```

1  #!/usr/bin/env python3
2
3
4  import Lab.utils
5
6  from . import DynamicSearch
7  from .AutoSchema import *
8  import collections
9  import psycopg2
10 import datetime
11 from Lab.view import View
12
13 import re
14 import Lab.utils
15 import collections
16
17
18 import psycopg2.extensions
19 import psycopg2.sql
20
21 import Lab.utils.psql_types
22
23
24 ▼ class Categories(SchemaTable):
25 ▼     def __init__(self, *args, **kwargs):
26         super().__init__(*args, **kwargs)
27         self.primary_key_name = f"Category_id"
28         #print(self.columns())
29
30 ▼     def columns(self):
31         row_type= collections.namedtuple("row type",'column_name data_type')
32         q1=Row_type('Category_id','bigserial')
33         q2=Row_type('Category_name','character varying')
34         result=(q1,q2)
35
36
37         return result
38
39 ▼     def addData(self, data: dict[collections.namedtuple] = None):
40         #print(data)
41
42         #print(self.columns())
43         #print(type(self.columns()))
44
45 ▼     if data is None:
46         #View.printInfo("None")
47         return Lab.utils.menuInput(self.addData, [a for a in self.columns()\
48             if a.column_name not in [f"{self.primary_key_name}"]])
49
50     NewName=next(a for a in data if a.column_name in ["Category_name"])
51     #print("Data name", data[NewName])
52
53
54     sql = """
55         INSERT INTO "Shop"."Categories" ("Category_name") VALUES (\'{data[NewName]}\');
56     """
57
58     with self.schema.dbconn.cursor() as dbcursor:
59         try:
60             View.printInfo(sql)
61
62             dbcursor.execute(sql)
63             self.schema.dbconn.commit()
64         except Exception as e:
65             self.schema.dbconn.rollback()
66             #print(f"Something went wrong: {e}")
67             View.printInfo(f"Something went wrong: {e}")
68             #print(type(e))
69             # raise e
70         else:
71             #print(f"dbcursor.rowcount} rows added")
72             #print(type({dbcursor.rowcount}))
73             View.printInfo(f"dbcursor.rowcount} rows added")
74
75
76
77     def editData(self, data: dict[collections.namedtuple] = None):
78         if data is None:
79             return Lab.utils.menuInput(self.editData, [a for a in self.columns() if a.column_name not in []])
80
81
82     NewName=next(a for a in data if a.column_name in ["Category_name"])
83     NewId=next(a for a in data if a.column_name in ["Category_id"])
84     #print("Data name", data[NewName])
85     #print("Data id", data[NewId])
86     sql = """
87         UPDATE "Shop"."Categories" SET "Category_name" = \'{data[NewName]}\'
88         WHERE "Category_id" = {data[NewId]};
89     """
89     View.printInfo(sql)
90
91     with self.schema.dbconn.cursor() as dbcursor:
92         try:
93             dbcursor.execute(sql)
94             self.schema.dbconn.commit()
95         except Exception as e:
96             self.schema.dbconn.rollback()
97             View.printInfo(f"Something went wrong: {e}")
98         else:
99             View.printInfo(f"dbcursor.rowcount} rows changed")

```

```

100
101     def removeData(self, rowid=None):
102         # rowid = click.prompt(f"{self.primary_key_name}", type=int)
103         if rowid is None:
104             return Lab.utils.menuInput(self.removeData, [a for a in self.columns()\
105                                         if a.column_name in [f"{self.primary_key_name}"]])
106
107     if isinstance(rowid, dict):
108         #printInfo(rowid)
109         #NewId=next(a for a in rowid if a.column_name in ["Category_id"])
110         #NewName=next(a for a in data if a.column_name in ["Category_name"])
111         rowid = rowid[next(a for a in rowid if a.column_name in ["Category_id"])]
112         #printInfo(rowid)
113
114     sql = f"""DELETE FROM "Shop"."Categories" WHERE "Category_id" = {rowid};"""
115
116     #sql = f"""DELETE FROM {self} WHERE "{self.primary_key_name}" = {rowid};"""
117     View.printInfo(sql)
118
119     with self.schema.dbconn.cursor() as dbcursor:
120         try:
121             dbcursor.execute(sql)
122             self.schema.dbconn.commit()
123         except Exception as e:
124             self.schema.dbconn.rollback()
125             View.printInfo(f"Something went wrong: {e}")
126         else:
127             View.printInfo(f"{dbcursor.rowcount} rows deleted")
128
129     def showData(self, sql=None):
130         # print(showDataCreator)
131         if sql is None:
132             sql = f"""SELECT * FROM "Shop"."Categories";"""
133             View.printInfo(sql)
134         return self.schema.showData(sql=sql)
135
136     def randomFill(self, instances: int = None, str_len: int = 10, sql_replace: str = None):
137         if instances is None:
138             instances = 100
139             return Lab.utils.menuInput(self.randomFill, [collections.namedtuple("instances", \
140                                         ["column_name", "data_type", "default"])(instances, "int", lambda: 100)])
141
142         if isinstance(instances, dict):
143             instances = instances[next(a for a in instances if a.column_name in ["instances"])]
144
145
146         sql = f"""
147             INSERT INTO "Shop"."Categories"("Category_name")
148                 SELECT
149                     substr(characters, (random() * length(characters) + 1)::integer, 10)
150                 FROM
151                     (VALUES('qwertyuiopasdfghijklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters),
152                         generate_series(1, {instances});
153
154 """
155
156         with self.schema.dbconn.cursor() as dbcursor:
157             try:
158                 View.printInfo(sql)
159                 t1 = datetime.datetime.now()
160                 dbcursor.execute(sql)
161                 t2 = datetime.datetime.now()
162                 self.schema.dbconn.commit()
163             except Exception as e:
164                 self.schema.dbconn.rollback()
165                 View.printInfo(f"Something went wrong: {e}")
166             else:
167                 View.printInfo(f"{self} {dbcursor.rowcount} rows added, execution time: {t2 - t1}")
168
169
170
171
172
173
174
175     class Manufacturer(SchemaTable):
176         def __init__(self, *args, **kwargs):
177             super().__init__(*args, **kwargs)
178             self.primary_key_name = f"Manufacturer_id"
179
180         def columns(self):
181             row_type= collections.namedtuple("row_type",'column_name data_type')
182             q1=row_type('Manufacturer_id','bigint')
183             q2=row_type('Manufacturer_name','character varying')
184             result=(q1,q2)
185             return result
186
187         def addData(self, data: dict[collections.namedtuple] = None):
188             #View.printInfo(data)
189             if data is None:
190                 #View.printInfo("None")
191                 return Lab.utils.menuInput(self.addData, [a for a in self.columns()\
192                                         if a.column_name not in [f"{self.primary_key_name}"]])
193
194             NewName=next(a for a in data if a.column_name in ["Manufacturer_name"])
195             #print("Data name", data[NewName])

```



```

297
298     """
299     with self.schema.dbconn.cursor() as dbcursor:
300         try:
301             View.printInfo(sql)
302             t1 = datetime.datetime.now()
303             dbcursor.execute(sql)
304             t2 = datetime.datetime.now()
305             self.schema.dbconn.commit()
306         except Exception as e:
307             self.schema.dbconn.rollback()
308             View.printInfo(f"Something went wrong: {e}")
309         else:
310             View.printInfo(f"{self} {dbcursor.rowcount} rows added, execution time: {t2 - t1}")
311
312
313
314 class Products(SchemaTable):
315     def __init__(self, *args, **kwargs):
316         super().__init__(*args, **kwargs)
317         self.primary_key_name = "Product_id"
318
319     def columns(self):
320         row_type = collections.namedtuple("row_type", 'column_name data_type')
321         q1 = row_type('Product_id', 'bigserial')
322         q2 = row_type('Manufacturer_id', 'bigserial')
323         q3 = row_type('Category_id', 'bigserial')
324         q4 = row_type('Product_name', 'character varying')
325         q5 = row_type('Price', 'money')
326         q6 = row_type('Amount', 'integer')
327         result = (q1, q2, q3, q4, q5, q6)
328         return result
329
330
331     def addData(self, data: dict[collections.namedtuple] = None):
332         #View.printInfo(data)
333         if data is None:
334             #View.printInfo("None")
335             return Lab.utils.menuInput(self.addData, [a for a in self.columns() \\
336                 if a.column_name not in [f"{self.primary_key_name}"]])
337
338         NewCategory_id = next(a for a in data if a.column_name in ["Category_id"])
339         NewManufacturer_id = next(a for a in data if a.column_name in ["Manufacturer_id"])
340         NewProduct_name = next(a for a in data if a.column_name in ["Product_name"])
341         NewPrice = next(a for a in data if a.column_name in ["Price"])
342         NewAmount = next(a for a in data if a.column_name in ["Amount"])
343
344
345         sql = """
346             INSERT INTO "Shop"."Products" ("Category_id", "Manufacturer_id", "Product_name", "Price", "Amount")
347             VALUES ({data[NewCategory_id]}, {data[NewManufacturer_id]}, '{data[NewProduct_name]}',
348             {data[NewPrice]}, {data[NewAmount]});
349         """
350
351         with self.schema.dbconn.cursor() as dbcursor:
352             try:
353
354                 View.printInfo(sql)
355                 dbcursor.execute(sql)
356                 self.schema.dbconn.commit()
357             except Exception as e:
358                 self.schema.dbconn.rollback()
359                 #print(f"Something went wrong: {e}")
360                 #raise(e)
361                 View.printInfo(f"Something went wrong: {e}")
362                 #print(type(e))
363                 # raise e
364             else:
365                 #print(f"{dbcursor.rowcount} rows added")
366                 #print(type(dbcursor.rowcount))
367                 View.printInfo(f"{dbcursor.rowcount} rows added")
368
369
370     def editData(self, data: dict[collections.namedtuple] = None):
371         if data is None:
372             return Lab.utils.menuInput(self.editData, [a for a in self.columns() if a.column_name not in []])
373
374         NewCategory_id = next(a for a in data if a.column_name in ["Category_id"])
375         NewManufacturer_id = next(a for a in data if a.column_name in ["Manufacturer_id"])
376         NewProduct_name = next(a for a in data if a.column_name in ["Product_name"])
377         NewPrice = next(a for a in data if a.column_name in ["Price"])
378         NewAmount = next(a for a in data if a.column_name in ["Amount"])
379         NewProduct_id = next(a for a in data if a.column_name in ["Product_id"])
380         #View.printInfoInfo("Data name", data[NewName])
381         #printInfo("Data id", data[NewId])
382         sql = """
383             UPDATE "Shop"."Products" SET "Category_id" = {data[NewCategory_id]},
384             "Manufacturer_id" = {data[NewManufacturer_id]},
385             "Product_name" = '{data[NewProduct_name]}', "Price" = {data[NewPrice]},
386             "Amount" = {data[NewAmount]} WHERE "Product_id" = {data[NewProduct_id]};
387         """
388         View.printInfo(sql)
389
390         with self.schema.dbconn.cursor() as dbcursor:
391             try:
392                 dbcursor.execute(sql)
393                 self.schema.dbconn.commit()
394             except Exception as e:
395                 self.schema.dbconn.rollback()
396                 View.printInfo(f"Something went wrong: {e}")
397             else:
398                 View.printInfo(f"{dbcursor.rowcount} rows changed")

```

```

398
399     def removeData(self, rowid=None):
400         # rowid = click.prompt(f"{self.primary_key_name}", type=int)
401         if rowid is None:
402             return Lab.utils.menuInput(self.removeData, [a for a in self.columns()\
403                                         if a.column_name in [f"{self.primary_key_name}"]])
404
405         if isinstance(rowid, dict):
406             #View.printInfo(rowid)
407             #NewId=next(a for a in rowid if a.column_name in ["Category_id"])
408             #NewName=next(a for a in data if a.column_name in ["Category_name"])
409             rowid = rowid[next(a for a in rowid if a.column_name in ["Product_id"])]
410             #View.printInfo(rowid)
411
412             sql = f"""DELETE FROM "Shop"."Products" WHERE "Product_id" = {rowid};"""
413
414             #sql = f"""DELETE FROM {self} WHERE "{self.primary_key_name}" = {rowid};"""
415             View.printInfo(sql)
416
417             with self.schema.dbconn.cursor() as dbcursor:
418                 try:
419                     dbcursor.execute(sql)
420                     self.schema.dbconn.commit()
421                 except Exception as e:
422                     self.schema.dbconn.rollback()
423                     View.printInfo(f"Something went wrong: {e}")
424                 else:
425                     View.printInfo(f"{dbcursor.rowcount} rows deleted")
426
427     def showData(self, sql=None):
428         # print(showDataCreator)
429         if sql is None:
430             sql = f"""SELECT * FROM "Shop"."Products";"""
431             View.printInfo(sql)
432         return self.schema.showData(sql=sql)
433
434     def randomFill(self, instances: int = None, str_len: int = 10, sql_replace: str = None):
435         if instances is None:
436             instances = 100
437             return Lab.utils.menuInput(self.randomFill, [collections.namedtuple("instances",\
438                                         ["column_name", "data_type", "default"])(instances, "int", lambda: 100)])
439
440         if isinstance(instances, dict):
441             instances = instances[next(a for a in instances if a.column_name in ["instances"])]
442
443
444         sql = f"""
445             INSERT INTO "Shop"."Products"("Category_id", "Manufacturer_id", "Product_name", "Price", "Amount")
446             SELECT
447
448                 (SELECT "Category id" FROM "Shop"."Categories" ORDER BY random()*q LIMIT 1),
449                 (SELECT "Manufacturer id" FROM "Shop"."Manufacturer" ORDER BY random()*q LIMIT 1),
450                 substr(characters, (random() * length(characters) + 1)::integer, 10),
451                 trunc(random() * 100)::int,
452                 trunc(random() * 100)::int
453                 FROM
454                 (VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters),
455                 generate_series(1, {instances}) as q;
456
457
458 """
459         #{instances}
460         with self.schema.dbconn.cursor() as dbcursor:
461             try:
462                 View.printInfo(sql)
463                 t1 = datetime.datetime.now()
464                 dbcursor.execute(sql)
465                 t2 = datetime.datetime.now()
466                 self.schema.dbconn.commit()
467             except Exception as e:
468                 self.schema.dbconn.rollback()
469                 View.printInfo(f"Something went wrong: {e}")
470             else:
471                 View.printInfo(f"{self} {dbcursor.rowcount} rows added, execution time: {t2 - t1}")
472
473
474     class User(SchemaTable):
475         def __init__(self, *args, **kwargs):
476             super().__init__(*args, **kwargs)
477             self.primary_key_name = f"User_data_id"
478
479         def columns(self):
480             row_type= collections.namedtuple("row_type",'column_name data_type')
481             q1=row_type('User_data_id','bigserial')
482             q2=row_type('Name','character varying')
483             q3=row_type('Surname','character varying')
484             q4=row_type('Patronymic','character varying')
485             q5=row_type('Email','character varying')
486             result=(q1,q2,q3,q4,q5)
487             return result
488
489         def addData(self, data: dict[collections.namedtuple] = None):
490             #View.printInfo(data)
491             if data is None:
492                 #View.printInfo("None")
493                 return Lab.utils.menuInput(self.addData, [a for a in self.columns()\
494                                         if a.column_name not in [f"{self.primary_key_name}"]])
495
496             NewName=next(a for a in data if a.column_name in ["Name"])
497             NewSurname=next(a for a in data if a.column_name in ["Surname"])
498             NewPatronymic=next(a for a in data if a.column_name in ["Patronymic"])

```

```

499     NewEmail=next(a for a in data if a.column_name in ["Email"])
500
501     sql = """
502         INSERT INTO "Shop"."User" ("Name", "Surname", "Patronymic", "Email")
503             VALUES (\'{data[NewName]}\', \'{data[NewSurname]}\', \'{data[NewPatronymic]}\', \'{data[NewEmail]}\');
504     """
505
506
507     with self.schema.dbconn.cursor() as dbcursor:
508         try:
509             View.printInfo(sql)
510             #print((psycopg2.extensions.AsIs(" ".join(map(lambda x: f'{x}', columns))), values))
511             dbcursor.execute(sql)
512             self.schema.dbconn.commit()
513         except Exception as e:
514             self.schema.dbconn.rollback()
515             #print(f"Something went wrong: {e}")
516             #raise(e)
517             View.printInfo(f"Something went wrong: {e}")
518             #print(type(e))
519             # raise e
520         else:
521             #print(f"{dbcursor.rowcount} rows added")
522             #print(type(dbcursor.rowcount))
523             View.printInfo(f"{dbcursor.rowcount} rows added")
524
525
526     def editData(self, data: dict[collections.namedtuple] = None):
527         if data is None:
528             return Lab.utils.menuInput(self.editData, [a for a in self.columns() if a.column_name not in []])
529
530         NewName=next(a for a in data if a.column_name in ["Name"])
531         NewSurname=next(a for a in data if a.column_name in ["Surname"])
532         NewPatronymic=next(a for a in data if a.column_name in ["Patronymic"])
533         NewEmail=next(a for a in data if a.column_name in ["Email"])
534         NewId=next(a for a in data if a.column_name in ["User_data_id"])
535         #View.printInfoInfo("Data_name", data[NewName])
536         #printInfo("Data_id", data[NewId])
537         sql = f"""UPDATE "Shop"."User" SET "Name" = \'{data[NewName]}\', "Surname" = \'{data[NewSurname]}\',
538             "Patronymic" = \'{data[NewPatronymic]}\', "Email" = \'{data[NewEmail]}\'
539             WHERE "User_data_id" = {data[NewId]};"""
540         View.printInfo(sql)
541
542         with self.schema.dbconn.cursor() as dbcursor:
543             try:
544                 dbcursor.execute(sql)
545                 self.schema.dbconn.commit()
546             except Exception as e:
547                 self.schema.dbconn.rollback()
548                 View.printInfo(f"Something went wrong: {e}")
549             else:
550                 View.printInfo(f"{dbcursor.rowcount} rows changed")
551
552
553     def removeData(self, rowid=None):
554         # rowid = click.prompt(f'{self.primary_key_name}', type=int)
555         if rowid is None:
556             return Lab.utils.menuInput(self.removeData, [a for a in self.columns()\n                if a.column_name in [f'{self.primary_key_name}']])
557
558
559         if isinstance(rowid, dict):
560             #View.printInfo(rowid)
561             #NewId=next(a for a in rowid if a.column_name in ["Category_id"])
562             #NewName=next(a for a in data if a.column_name in ["Category_name"])
563             rowid = rowid[next(a for a in rowid if a.column_name in ["User_data_id"])]
564             #View.printInfo(rowid)
565
566         sql = """DELETE FROM "Shop"."User" WHERE "User_data_id" = {rowid};"""
567
568         #sql = """DELETE FROM {self} WHERE "{self.primary_key_name}" = {rowid};"""
569         View.printInfo(sql)
570
571         with self.schema.dbconn.cursor() as dbcursor:
572             try:
573                 dbcursor.execute(sql)
574                 self.schema.dbconn.commit()
575             except Exception as e:
576                 self.schema.dbconn.rollback()
577                 View.printInfo(f"Something went wrong: {e}")
578             else:
579                 View.printInfo(f"{dbcursor.rowcount} rows deleted")
580
581     def showData(self, sql=None):
582         # print(showDataCreator)
583         if sql is None:
584             sql = """SELECT * FROM "Shop"."User";"""
585             View.printInfo(sql)
586         return self.schema.showData(sql=sql)
587
588     def randomFill(self, instances: int = None, str_len: int = 10, sql_replace: str = None):
589         if instances is None:
590             #instances = 100
591             return Lab.utils.menuInput(self.randomFill, [collections.namedtuple("instances",\n                ["column_name", "data_type", "default"])(instances, "int", lambda: 100)])
592
593         if isinstance(instances, dict):
594             instances = instances[next(a for a in instances if a.column_name in ["instances"])]
```

```

598     sql = """
599     INSERT INTO "Shop"."User"("Name", "Surname", "Patronymic", "Email")
600         SELECT
601             substr(characters, (random() * length(characters) + 1)::integer, 10),
602             substr(characters, (random() * length(characters) + 1)::integer, 10),
603             substr(characters, (random() * length(characters) + 1)::integer, 10),
604             substr(characters, (random() * length(characters) + 1)::integer, 10)
605         FROM
606             (VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters);
607
608
609     """
610
611     with self.schema.dbconn.cursor() as dbcursor:
612         try:
613             View.printInfo(sql)
614             t1 = datetime.datetime.now()
615             dbcursor.execute(sql)
616             t2 = datetime.datetime.now()
617             self.schema.dbconn.commit()
618         except Exception as e:
619             self.schema.dbconn.rollback()
620             View.printInfo(f"Something went wrong: {e}")
621         else:
622             View.printInfo(f"{self} {dbcursor.rowcount} rows added, execution time: {t2 - t1}")
623
624
625     class Order(SchemaTable):
626         def __init__(self, *args, **kwargs):
627             super().__init__(*args, **kwargs)
628             self.primary_key_name = "Order_id"
629
630         def columns(self):
631             row_type = collections.namedtuple("row_type", "column_name data_type")
632             q1 = row_type("Order_id", "bigserial")
633             q2 = row_type("User_data_id", "bigserial")
634             result = (q1, q2)
635             return result
636
637         def addData(self, data: dict[collections.namedtuple] = None):
638             #View.printInfo(data)
639             if data is None:
640                 #View.printInfo("None")
641                 return Lab.utils.menuInput(self.addData, [a for a in self.columns()\
642                     if a.column_name not in [f"{self.primary_key_name}"]])
643
644             NewUser_data_id = next(a for a in data if a.column_name in ["User_data_id"])
645
646
647             sql = """
648                 INSERT INTO "Shop"."Order" ("User_data_id") VALUES ({data[NewUser_data_id]});
649             """
650
651             with self.schema.dbconn.cursor() as dbcursor:
652                 try:
653                     View.printInfo(sql)
654                     #print((psycopg2.extensions.AsIs(" ".join(map(lambda x: f"'{x}'", columns))), values))
655                     dbcursor.execute(sql)
656                     self.schema.dbconn.commit()
657                 except Exception as e:
658                     self.schema.dbconn.rollback()
659                     #print(f"Something went wrong: {e}")
660                     View.printInfo(f"Something went wrong: {e}")
661                     #print(type(e))
662                     # raise e
663                 else:
664                     #print(f"{dbcursor.rowcount} rows added")
665                     #print(type({dbcursor.rowcount}))
666                     View.printInfo(f"{dbcursor.rowcount} rows added")
667
668
669         def editData(self, data: dict[collections.namedtuple] = None):
670             if data is None:
671                 return Lab.utils.menuInput(self.editData, [a for a in self.columns() if a.column_name not in []])
672
673
674             NewUser_data_id = next(a for a in data if a.column_name in ["User_data_id"])
675             NewOrder_id = next(a for a in data if a.column_name in ["Order_id"])
676
677             sql = f"""UPDATE "Shop"."Order" SET "User_data_id" = {data[NewUser_data_id]}\
678             WHERE "Order_id" = {data[NewOrder_id]};"""
679             View.printInfo(sql)
680
681             with self.schema.dbconn.cursor() as dbcursor:
682                 try:
683                     dbcursor.execute(sql)
684                     self.schema.dbconn.commit()
685                 except Exception as e:
686                     self.schema.dbconn.rollback()
687                     View.printInfo(f"Something went wrong: {e}")
688                 else:
689                     View.printInfo(f"{dbcursor.rowcount} rows changed")
690
691
692
693         def removeData(self, rowid=None):
694             # rowid = click.prompt(f"{self.primary_key_name}", type=int)
695             if rowid is None:
696                 return Lab.utils.menuInput(self.removeData, [a for a in self.columns()\
697                     if a.column_name in [f"{self.primary_key_name}"]])
698

```

```

699     if isinstance(rowid, dict):
700         rowid = rowid[next(a for a in rowid if a.column_name in ["Order_id"])]
701
702
703
704     sql = """DELETE FROM "Shop"."Order" WHERE "Order_id" = {rowid};"""
705
706     #sql = f"""DELETE FROM {self} WHERE "{self.primary_key_name}" = {rowid};"""
707     View.printInfo(sql)
708
709     with self.schema.dbconn.cursor() as dbcursor:
710         try:
711             dbcursor.execute(sql)
712             self.schema.dbconn.commit()
713         except Exception as e:
714             self.schema.dbconn.rollback()
715             View.printInfo(f"Something went wrong: {e}")
716         else:
717             View.printInfo(f"{dbcursor.rowcount} rows deleted")
718
719     def showData(self, sql=None):
720         # print(showDataCreator)
721         if sql is None:
722             sql = """SELECT * FROM "Shop"."Order";"""
723             View.printInfo(sql)
724         return self.schema.showData(sql=sql)
725
726     def randomFill(self, instances: int = None, str_len: int = 10, sql_replace: str = None):
727         if instances is None:
728             instances = 100
729             return Lab.utils.menuInput(self.randomFill, [collections.namedtuple("instances",\
730                 ["column_name", "data_type", "default"])(instances, "int", lambda: 100)])
731
732         if isinstance(instances, dict):
733             instances = instances[next(a for a in instances if a.column_name in ["instances"])]
734
735
736         sql = """
737             INSERT INTO "Shop"."Order"("User_data_id")
738             SELECT
739                 (SELECT "User_data_id" FROM "Shop"."User" ORDER BY random()*q LIMIT 1)
740             FROM
741                 generate_series(1, {instances}) as q;
742         """
743         with self.schema.dbconn.cursor() as dbcursor:
744             try:
745                 View.printInfo(sql)
746                 t1 = datetime.datetime.now()
747
748                 dbcursor.execute(sql)
749                 t2 = datetime.datetime.now()
750                 self.schema.dbconn.commit()
751             except Exception as e:
752                 self.schema.dbconn.rollback()
753                 View.printInfo(f"Something went wrong: {e}")
754             else:
755                 View.printInfo(f"{self} {dbcursor.rowcount} rows added, execution time: {t2 - t1}")
756
757
758     class Ordered_product(SchemaTable):
759         def __init__(self, *args, **kwargs):
760             super().__init__(*args, **kwargs)
761             self.primary_key_name = "Ordered_product_id"
762
763         def columns(self):
764             row_type= collections.namedtuple("row_type",'column_name data_type')
765             q1=row_type('Ordered_product_id','bigserial')
766             q2=row_type('Product_id','bigserial')
767             q3=row_type('Ordered_amount','integer')
768             q4=row_type('Order_id','bigserial')
769             result=(q1,q2,q3,q4)
770             return result
771
772         def addData(self, data: dict[collections.namedtuple] = None):
773             #View.printInfo(data)
774             if data is None:
775                 #View.printInfo("None")
776                 return Lab.utils.menuInput(self.addData, [a for a in self.columns()\
777                     if a.column_name not in [f"{self.primary_key_name}"]])
778
779             NewProduct_id=next(a for a in data if a.column_name in ["Product_id"])
780             NewOrdered_amount=next(a for a in data if a.column_name in ["Ordered_amount"])
781             NewOrder_id=next(a for a in data if a.column_name in ["Order_id"])
782
783
784         sql = """
785             INSERT INTO "Shop"."Ordered_product" ("Product_id", "Ordered_amount", "Order_id")
786             VALUES ({data[NewProduct_id]}, {data[NewOrdered_amount]}, {data[NewOrder_id]});
787         """
788
789         with self.schema.dbconn.cursor() as dbcursor:
790             try:
791                 View.printInfo(sql)
792
793                 dbcursor.execute(sql)
794                 self.schema.dbconn.commit()
795             except Exception as e:
796

```

```
797         self.schema.dbconn.rollback()
798         View.printInfo(f"Something went wrong: {e}")
799
800     else:
801         View.printInfo(f"{dbcursor.rowcount} rows added")
802
803
804
805
806     def editData(self, data: dict[collections.namedtuple] = None):
807         if data is None:
808             return Lab.utils.menuInput(self.editData, [a for a in self.columns() if a.column_name not in []])
809
810         NewProduct_id=next(a for a in data if a.column_name in ["Product_id"])
811         NewOrdered_amount=next(a for a in data if a.column_name in ["Ordered_amount"])
812         NewOrder_id=next(a for a in data if a.column_name in ["Order_id"])
813         NewOrdered_product_id=next(a for a in data if a.column_name in ["Ordered_product_id"])
814         #View.printInfo("Data name", data[NewName])
815         #printInfo("Data id", data[NewId])
816         sql = f"""UPDATE "Shop"."Ordered_product" SET "Product_id" = {data[NewProduct_id]},
817         "Ordered amount" = {data[NewOrdered_amount]},
818         "Order id" = {data[NewOrder_id]} WHERE "Ordered_product_id" = {data[NewOrdered_product_id]};"""
819         View.printInfo(sql)
820
821         with self.schema.dbconn.cursor() as dbcursor:
822             try:
823                 dbcursor.execute(sql)
824                 self.schema.dbconn.commit()
825             except Exception as e:
826                 self.schema.dbconn.rollback()
827                 View.printInfo(f"Something went wrong: {e}")
828             else:
829                 View.printInfo(f"{dbcursor.rowcount} rows changed")
830
831
832     def removeData(self, rowid=None):
833         # rowid = click.prompt(f"{self.primary_key_name}", type=int)
834         if rowid is None:
835             return Lab.utils.menuInput(self.removeData, [a for a in self.columns() \
836                 if a.column_name in [f"{self.primary_key_name}"]])
837
838         if isinstance(rowid, dict):
839             #View.printInfo(rowid)
840             #NewId=next(a for a in rowid if a.column_name in ["Category_id"])
841             #NewName=next(a for a in data if a.column_name in ["Category_name"])
842             rowid = rowid[next(a for a in rowid if a.column_name in ["Ordered_product_id"])]
843             #View.printInfo(rowid)
844
845         sql = f"""DELETE FROM "Shop"."Ordered_product" WHERE "Ordered_product_id" = {rowid};"""
846
847
848         View.printInfo(sql)
849
850         with self.schema.dbconn.cursor() as dbcursor:
851             try:
852                 dbcursor.execute(sql)
853                 self.schema.dbconn.commit()
854             except Exception as e:
855                 self.schema.dbconn.rollback()
856                 View.printInfo(f"Something went wrong: {e}")
857             else:
858                 View.printInfo(f"{dbcursor.rowcount} rows deleted")
859
860     def showData(self, sql=None):
861         # print(showDataCreator)
862         if sql is None:
863             sql = f"""SELECT * FROM "Shop"."Ordered_product";"""
864             View.printInfo(sql)
865         return self.schema.showData(sql=sql)
866
867     def randomFill(self, instances: int = None, str_len: int = 10, sql_replace: str = None):
868         if instances is None:
869             #instances = 100
870             return Lab.utils.menuInput(self.randomFill, [collections.namedtuple("instances", \
871                 ["column_name", "data_type", "default"])(instances, "int", lambda: 100)])
872
873         if isinstance(instances, dict):
874             instances = instances[next(a for a in instances if a.column_name in ["instances"])]
875
876
877         sql = f"""
878             INSERT INTO "Shop"."Ordered_product"("Product_id", "Ordered_amount", "Order_id")
879             SELECT
880
881                 (SELECT "Product_id" FROM "Shop"."Products" ORDER BY random()*q LIMIT 1),
882                 trunc(random() *100)::int,
883                 (SELECT "Order_id" FROM "Shop"."Order" ORDER BY random()*q LIMIT 1)
884
885             FROM
886                 generate_series(1,{instances} ) as q;
887
888
889
890         #{instances}
891         with self.schema.dbconn.cursor() as dbcursor:
892             try:
893                 View.printInfo(sql)
894                 t1 = datetime.datetime.now()
895                 dbcursor.execute(sql)
896                 t2 = datetime.datetime.now()
897             except Exception as e:
```

```

898         self.schema.dbconn.commit()
899     except Exception as e:
900         self.schema.dbconn.rollback()
901         View.printInfo(f"Something went wrong: {e}")
902     else:
903         View.printInfo(f"\n{self} {dbcursor.rowcount} rows added, execution time: {t2 - t1}\n")
904
905
906 ▼ class Shop(Schema):
907     def __init__(self, *args, **kwargs):
908         super().__init__(*args, **kwargs)
909         self.dynamicsearch = {a.name: a for a in [DynamicSearch.CategoriesProductsDynamicSearch(self), \
910                                         DynamicSearch.ManufacturerProductsDynamicSearch(self), \
911                                         DynamicSearch.UserOrderedProductsDynamicSearch(self),]}
912         # self.reoverride()
913
914 ▼ def reoverride(self):
915     # Table override
916     # self.tables.Loan = LoanTable(self, f"Loan")
917     # print(f"self")
918     self.tables.Categories = Categories(self, f"Categories")
919     self.tables.Manufacturer = Manufacturer(self, f"Manufacturer")
920     self.tables.Products = Products(self, f"Products")
921     self.tables.User = User(self, f"User")
922     self.tables.Order = Order(self, f"Order")
923     self.tables.Ordered_product = Ordered_product(self, f"Ordered_product")
924
925 ▼ def reinit(self):
926     # sql = """
927     #   SELECT table name FROM information_schema.tables
928     #   WHERE table_schema = '{self}';
929
930     # """
931     with self.dbconn.cursor() as dbcursor:
932         # dbcursor.execute(sql)
933         for a in self.refresh_tables(): # tuple(dbcursor.fetchall()):
934             q = f"""DROP TABLE IF EXISTS {a} CASCADE;"""
935             # print(q)
936             dbcursor.execute(q)
937
938     tables = [
939         f"""CREATE SCHEMA IF NOT EXISTS "{self}";""",
940         f"""CREATE TABLE IF NOT EXISTS "{self}"."Categories"(
941             "Category_id" bigserial,
942             "Category_name" character varying(70) NOT NULL,
943             PRIMARY KEY ("Category_id")
944         );
945
946         f"""CREATE TABLE IF NOT EXISTS "{self}"."Manufacturer"(
947             "Manufacturer_id" bigserial,
948             "Manufacturer_name" character varying(70) NOT NULL,
949             PRIMARY KEY ("Manufacturer_id")
950         );
951
952         f"""CREATE TABLE IF NOT EXISTS "{self}"."Products"(
953             "Product_id" bigserial,
954             "Category_id" bigint NOT NULL,
955             "Manufacturer_id" bigint NOT NULL,
956             "Product_name" character varying(70) NOT NULL,
957             "Price" money NOT NULL,
958             "Amount" integer NOT NULL,
959             PRIMARY KEY ("Product_id"),
960             FOREIGN KEY ("Category_id") REFERENCES "{self}"."Categories"("Category_id"),
961             FOREIGN KEY ("Manufacturer_id") REFERENCES "{self}"."Manufacturer"("Manufacturer_id")
962         );
963
964         f"""CREATE TABLE IF NOT EXISTS "{self}"."User"(
965             "User_data_id" bigserial,
966             "Name" character varying(40) NOT NULL,
967             "Surname" character varying(40) NOT NULL,
968             "Patronymic" character varying(50) NOT NULL,
969             "Email" character varying(255) NOT NULL,
970             PRIMARY KEY ("User_data_id")
971         );
972
973         f"""CREATE TABLE IF NOT EXISTS "{self}"."Order"(
974             "Order_id" bigserial,
975             "User_data_id" bigint NOT NULL,
976             PRIMARY KEY ("Order_id"),
977             FOREIGN KEY ("User_data_id") REFERENCES "{self}"."User"("User_data_id")
978         );
979
980         f"""CREATE TABLE IF NOT EXISTS "{self}"."Ordered_product"(
981             "Ordered_product_id" bigserial,
982             "Product_id" bigint NOT NULL,
983             "Ordered_amount" integer NOT NULL,
984             "Order_id" bigint NOT NULL,
985             PRIMARY KEY ("Ordered_product_id"),
986             FOREIGN KEY ("Product_id") REFERENCES "{self}"."Products"("Product_id"),
987             FOREIGN KEY ("Order_id") REFERENCES "{self}"."Order"("Order_id")
988         );
989
990     ]
991
992     with self.dbconn.cursor() as dbcursor:
993         for a in tables:
994             dbcursor.execute(a)
995
996     self.dbconn.commit()
997
998     self.refresh_tables()

```

```
999     # self.reoverride()
1000
1001     def randomFill(self):
1002         self.tables.Categories.randomFill(1_000)
1003         self.tables.Manufacturer.randomFill(1_000)
1004         self.tables.Products.randomFill(1_000)
1005         self.tables.User.randomFill(1_000)
1006         self.tables.Order.randomFill(1_000)
1007         self.tables.Ordered_product.randomFill(1_000)
1008
1009
1010     def _test():
1011         pass
1012
1013
1014     if __name__ == "__main__":
1015         _test()
1016
1017
```

DynamicSearch.py

```

1  #!/usr/bin/env python
2  import itertools
3  import pprint
4  from Lab.view import View
5  from .dynamicsearch import *
6
7  all_ = ["CategoriesProductsDynamicSearch", "ManufacturerProductsDynamicSearch", \
8  "UserOrderedProductsDynamicSearch"]
9
10
11
12
13 ▼ class CategoriesProductsDynamicSearch(DynamicSearchBase):
14 ▼   def __init__(self, *args, **kwargs):
15     super().__init__(*args, **kwargs)
16     self.name: str = "CategoriesProducts"
17     self.search: dict[self.SearchCriterias[CompareConstant]] = {
18       "Category_name": SearchCriterias(f'"a"."Category_name'", f"Category_name", "varchar"),
19       "Price": SearchCriterias(f'"b"."Price"', f"Price", "money"),
20       "Amount": SearchCriterias(f'"b"."Amount"', f"Amount", "integer"),
21     }
22
23
24   @property
25   def sql(self):
26     where = self.where
27     sql = f"""
28       SELECT
29         "a"."Category_name",
30         "a"."Category_id",
31         "b"."Product_id",
32         "b"."Product_name",
33         "b"."Price",
34         "b"."Amount"
35     FROM
36       "{self.schema}"."Categories" as "a"
37       INNER JOIN "{self.schema}"."Products" as "b"
38         ON "a"."Category_id" = "b"."Category_id"
39
40       {f'''WHERE
41         {where};''' if where else f";"}
42     """
43
44   View.printInfo(sql)
45   return sql
46
47 ▼ class ManufacturerProductsDynamicSearch(DynamicSearchBase):
48 ▼   def __init__(self, *args, **kwargs):
49     super().__init__(*args, **kwargs)
50     self.name: str = "ManufacturerProducts"
51     self.search: dict[self.SearchCriterias[CompareConstant]] = {
52       "Manufacturer_name": SearchCriterias(f'"a"."Manufacturer_name'", f"Manufacturer_name", "varchar"),
53       "Price": SearchCriterias(f'"b"."Price"', f"Price", "money"),
54       "Product_name": SearchCriterias(f'"b"."Product_name'", f"Product_name", "varchar"),
55     }
56
57
58   @property
59   def sql(self):
60     where = self.where
61     sql = f"""
62       SELECT
63         "a"."Manufacturer_name",
64         "a"."Manufacturer_id",
65         "b"."Product_id",
66         "b"."Product_name",
67         "b"."Price",
68         "b"."Amount"
69     FROM
70       "{self.schema}"."Manufacturer" as "a"
71       INNER JOIN "{self.schema}"."Products" as "b"
72         ON "a"."Manufacturer_id" = "b"."Manufacturer_id"
73
74       {f'''WHERE
75         {where};''' if where else f";"}
76     """
77
78   View.printInfo(sql)
79   return sql
80
81
82 class UserOrderedProductsDynamicSearch(DynamicSearchBase):
83   def __init__(self, *args, **kwargs):
84     super().__init__(*args, **kwargs)
85     self.name: str = "UserOrderedProducts"
86     self.search: dict[self.SearchCriterias[CompareConstant]] = {
87       "Name": SearchCriterias(f'"c"."Name"', f"Name", "varchar"),
88       "Surname": SearchCriterias(f'"c"."Surname"', f"Surname", "varchar"),
89       "Ordered_amount": SearchCriterias(f'"a"."Ordered_amount"', f"Ordered_amount", "integer"),
90     }
91
92
93   @property
94   def sql(self):
95     where = self.where
96     sql = f"""
97       SELECT
98         "a"."Ordered_product_id",
99         "a"."Ordered_amount",
100        "a"."Order_id",
101        "c"."Name",
102        "c"."Surname",
103        "c"."Patronymic",

```

```

104         "d"."Product_name"
105     FROM
106     "{self.schema}"."Ordered_product" as "a"
107     INNER JOIN "{self.schema}"."Order" as "b"
108     ON "a"."Order_id" = "b"."Order_id"
109     INNER JOIN "{self.schema}"."User" as "c"
110     ON "b"."User_data_id" = "c"."User_data_id"
111     INNER JOIN "{self.schema}"."Products" as "d"
112     ON "a"."Product_id" = "d"."Product_id"
113     {f'` WHERE
114     {where};`' if where else f";`}
115     """
116     View.printInfo(sql)
117     return sql
118
119
120 def _test():
121     pass
122
123
124 if __name__ == "__main__":
125     _test()
126

```

dynamicsearch.py

```

1 #!/usr/bin/env python
2 import Lab.utils
3 import datetime
4 import itertools
5 import collections
6 import Lab.utils.psql_types
7
8 all_ = [
9     "CompareConstant",
10    "SearchCriterias",
11    "SelectCompositor",
12    "DynamicSearchBase",
13 ]
14
15
16 class CompareConstant(object):
17     def __init__(self, psql_type, comparator=None, constant=None):
18         super().__init__()
19         self.comparator = comparator
20         self._constant = None
21         self._psql_type = psql_type
22
23     def __str__(self):
24         if self.isIgnored:
25             return f"""ignored"""
26         # if isinstance(self.constant, str) else self.constant
27         return f"""{self.comparator} {self.constant}::{self._psql_type}"""
28
29     def __repr__(self):
30         return f"""{type(self).__name__}(comparator={self.comparator}, constant={self.constant})"""
31
32     def reset(self):
33         self.comparator = None
34         self.setNull()
35
36     def setNull(self):
37         self.constant = None
38
39     def setConstant(self, constant=None):
40         if constant is None:
41             return Lab.utils.menuInput(self.setConstant, [collections.namedtuple("instances", \
42                 ["column_name", "data_type", "default"])(self._psql_type, self._psql_type, lambda: None)])
43         else:
44             self.constant = constant[next(a for a in constant if a.column_name in [self._psql_type])]
45
46
47     @property
48     def isIgnored(self):
49         return self.comparator is None
50
51     @property
52     def psql_type(self):

```

```

53     return self._pgsql_type
54
55     @property
56     def constant(self):
57         if isinstance(self.constant, (str, datetime.datetime,)):
58             return f'{self.constant}'
59         elif self.constant is None:
60             return f'NULL'
61         # print(type(self.constant))
62         return self._constant
63
64     @constant.setter
65     def constant(self, value):
66         self._constant = value
67
68     def _lt(self):
69         self.comparator = "<"
70
71     def _le(self):
72         self.comparator = "<="
73
74     def _eq(self):
75         self.comparator = "="
76
77     def _ne(self):
78         self.comparator = "!="
79
80     def _ge(self):
81         self.comparator = ">="
82
83     def _gt(self):
84         self.comparator = ">"
85
86     def _like(self):
87         self.comparator = "LIKE"
88
89     @property
90     def prompt(self) -> str:
91         return f"Criteria editor: {self}"
92
93     @property
94     def _lab_console_interface_(self):
95         result = Lab.utils.LabConsoleInterface({
96             "ignore": self.reset,
97             "<": self._lt,
98             "<=". self._le,
99             "=": self._eq,
100            "!=": self._ne,
101            ">=". self._ge,
102            ">": self._gt,
103            "LIKE": self._like,
104            "# IS": lambda: setattr(self, "comparator", "IS"),
105            "# IS NOT": lambda: setattr(self, "comparator", "IS NOT"),
106            "set NULL": self.setNull,
107            "set constant": self.setConstant,
108            "# moar": lambda: self,
109            "return": lambda: Lab.utils.menuReturn(f"[User] menu return"),
110        }, prompt=self.prompt)
111        return result
112
113
114 class SearchCriterias(list):
115     def __init__(self, psql_mapping: str, psql_name: str, psql_type: str, *args, **kwargs):
116         super().__init__(*args, **kwargs)
117         self._pgsql_mapping = psql_mapping
118         self._pgsql_name = psql_name
119         self._pgsql_type = psql_type
120
121     @property
122     def psql_mapping(self):
123         return self._pgsql_mapping
124
125     @property
126     def psql_name(self):
127         return self._pgsql_name
128
129     @property
130     def psql_type(self):
131         return self._pgsql_type
132
133     def reset(self):
134         self.clear()
135
136     def append(self):
137         # if isinstance(obj, CompareConstant):
138         #     return super().append(obj)
139         # elif obj is None:
140         #     return super().append(obj)
141         # raise TypeError(f"{{type(obj)}} is invalid")CompareConstant(self.psql_type)
142         # q = CompareConstant(self.psql_type)
143
144         try:
145             next(a for a, b in enumerate(self) if b.isIgnored)
146         except StopIteration:
147             super().append(CompareConstant(self.psql_type))
148
149         return self
150
151     def gen_sql(self):
152         result = f"""\{ AND ".join(f'{self._pgsql_mapping} {a}' for a in self if not a.isIgnored)}"""
153         # print(f'{result}')
154         if result:
155             result = f'{{result}}'

```

```

156     return result
157
158     @property
159     def sql(self):
160         return self.gen_sql()
161
162     def __format__(self, format_=None):
163         if format_ == "v":
164             return f"{{list(filter(lambda x: not (x.isIgnored), {self}))}}"
165         elif format_ == "sql":
166             return self.gen_sql()
167         elif format_ == "pre":
168             result = f"""{ " AND ".join(f"{a}" for a in self if not a.isIgnored)}"""
169             if result:
170                 return result
171             return f"ignored"
172         return super().__format__(format_)
173
174     # def append_if_needed(self):
175     #     @property
176     #     def __lab_console_interface__(self):
177     #         result = Lab.utils.LabConsoleInterface()
178     #         result.update({f"Property {a} {b}": (lambda x: lambda: x)(b) for a, b in enumerate(self, 1)})
179     #         result.prompt = f"{self}"
180     #         return result
181
182
183     class SelectCompositor(object):
184         def __init__(self, search_criterias, table):
185             super().__init__()
186             self._search_criterias: SearchCriterias[CompareConstant] = search_criterias
187             self._table = table
188             self._search_criterias.append()
189
190         @property
191         def table(self):
192             return self._table
193
194         @property
195         def search_criterias(self):
196             return self._search_criterias
197
198         @property
199         def prompt(self):
200             return f"{{self.table}} {self.search_criterias:pre} select criterias:"
201
202
203         @property
204         def __lab_console_interface__(self):
205             try:
206                 self._search_criterias.append()
207
208                 result = Lab.utils.LabConsoleInterface({
209                     **{f"Property {a} {b}": (lambda x: lambda: x)(b) for a, b in enumerate(self._search_criterias, 1)},
210                     # "new criterias": lambda: self._search_criterias[self._table].append(),
211                     "return": lambda: Lab.utils.menuReturn(f"User menu return"),
212                 }, prompt=self.prompt)
213                 return result
214             except Exception as e:
215                 print(e)
216
217         def __bool__(self):
218             return bool(self._search_criterias)
219
220         # def reset(self):
221         #     self._search_criterias.reset()
222
223         # def __format__(self, *args, **kwargs):
224         #     return self._search_criterias.__format__(*args, **kwargs)
225
226     class DynamicSearchBase(object):
227         def __init__(self, schema):
228             super().__init__()
229             self.name = type(self).__name__
230             self.schema = schema
231             self._search: dict[SelectCompositor] = dict()
232             # self.selectcompositors = tuple()
233
234         @property
235         def search(self) -> dict[SelectCompositor]:
236             return self._search
237
238         @search.setter
239         def search(self, value: dict):
240             self._search = dict(itertools.starmap(lambda key, value: (key, SelectCompositor(value, key)), value.items()))
241
242         def execute(self) -> Lab.utils.TablePrint:
243             return self.schema.showData(sql=self.sql)
244
245         def reset(self) -> None:
246             for a in self._search.values():
247                 a._search_criterias.reset()
248
249         @property
250         def where(self) -> str:
251             newline = " AND \n"
252             return newline.join(f"{{a.search_criterias:sql}}" for a in self._search.values() if f"{{a.search_criterias:sql}}")
253
254         @property
255         def sql(self) -> str:
256             raise NotImplementedError(f"Need to override")
257

```

```

258
259     @property
260     def prompt(self):
261         newline = f"\n"
262         return f"""{self.name} dynamic search interface\n{newline.join(f'{"{a}" {b.search_criterias:pre}}'\
263             for a, b in self.search.items())}"""
264
265     @property
266     def __lab_console_interface__(self):
267         try:
268             result = Lab.utils.LabConsoleInterface({
269                 # **{f'{a}': (lambda x: lambda: SelectCompositor(self.search[x], x))(a) for a in self.search},
270                 **{a: (lambda x: lambda: x)(b) for a, b in self.search.items()},
271                 f'execute': self.execute,
272                 f'sql': lambda: print(self.sql),
273                 f'reset': self.reset,
274                 f'return': lambda: Lab.utils.menuReturn(f"User menu return"),
275             }, prompt=self.prompt)
276             return result
277         except Exception as e:
278             print(e)
279
280
281     def test():
282         pass
283
284
285     if __name__ == "__main__":
286         _test()
287

```

Короткий опис функцій

Файл Schema.py складається з класів Categories, Manufacturer, Products, User, Order, Ordered_product та Shop

Класи Categories, Manufacturer, Products, User, Order, Ordered_product

відповідно мають в своєму складі функції для роботи з відповідними таблицями
що називаються так само як і класи, 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

