

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

Заварін В. О.

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.

4. 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

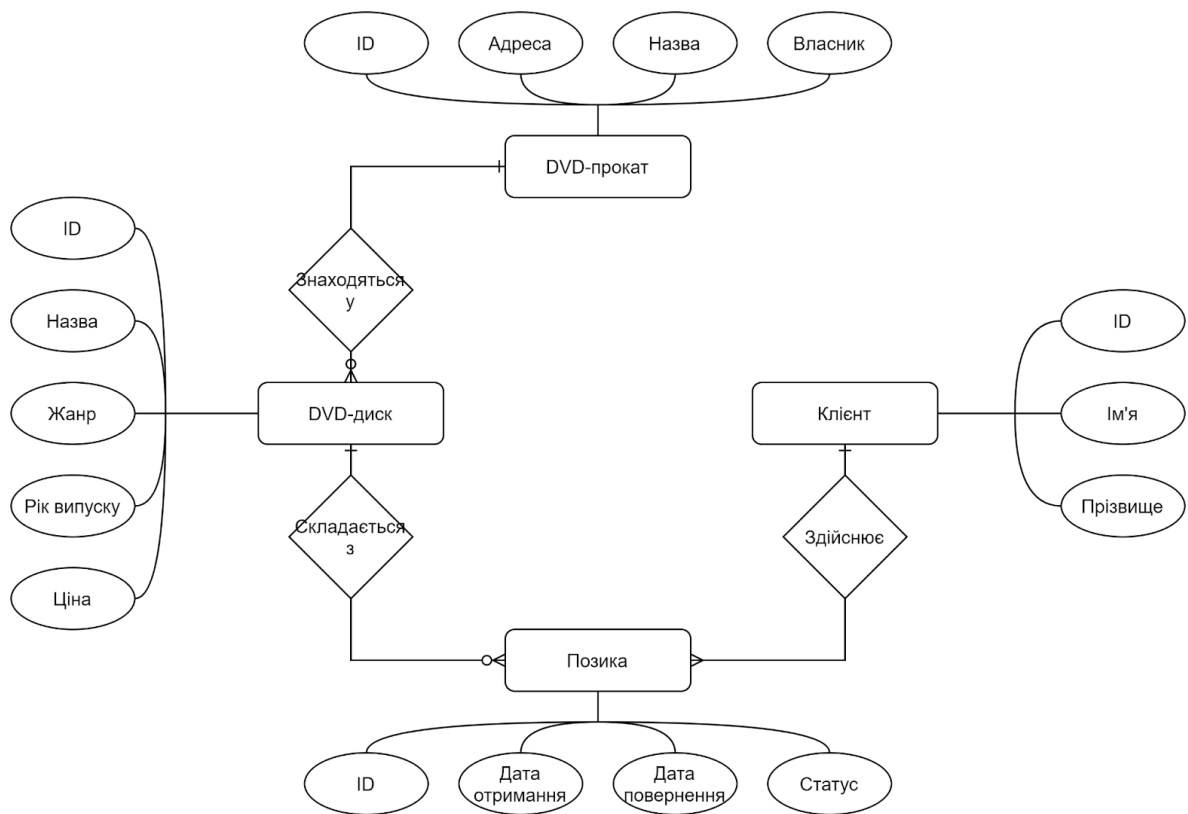


Рис. 1 - Концептуальна модель предметної області “DVD-прокат”

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

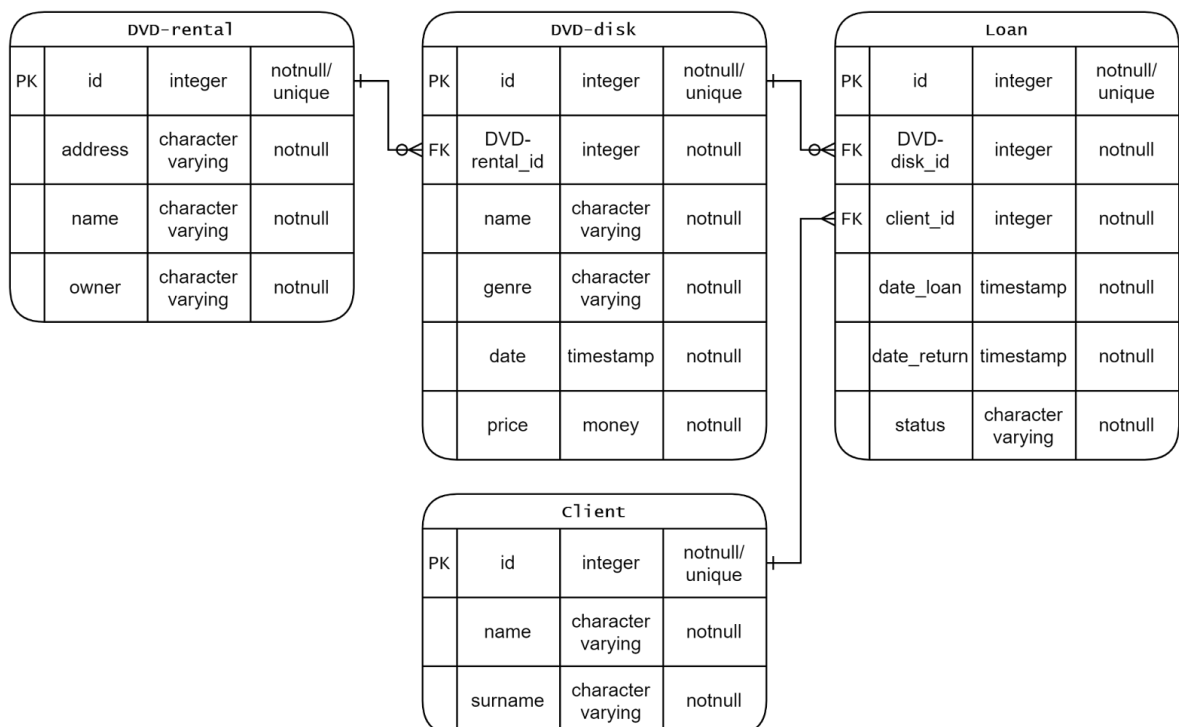


Рис. 2 – Логічна модель бази даних

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».

```
lvizav@vizav-virtualbox lab2db1$ tree
.
├── Lab
│   ├── controller
│   │   ├── Controller.py
│   │   ├── __init__.py
│   │   └── __pycache__
│   │       ├── Controller.cpython-310.pyc
│   │       ├── Controller.cpython-39.pyc
│   │       ├── __init__.cpython-310.pyc
│   │       └── __init__.cpython-39.pyc
│   ├── __init__.py
│   └── model
│       ├── AutoSchema.py
│       ├── dynamicsearch.py
│       ├── DynamicSearch.py
│       ├── __init__.py
│       ├── __pycache__
│       │   ├── AutoSchema.cpython-310.pyc
│       │   ├── AutoSchema.cpython-39.pyc
│       │   ├── dynamicsearch.cpython-310.pyc
│       │   ├── DynamicSearch.cpython-310.pyc
│       │   ├── dynamicsearch.cpython-39.pyc
│       │   ├── DynamicSearch.cpython-39.pyc
│       │   ├── __init__.cpython-310.pyc
│       │   └── __init__.cpython-39.pyc
│       ├── psycopg.cpython-39.pyc
│       ├── Schema.cpython-310.pyc
│       └── Schema.cpython-39.pyc
│       └── Schema.py
│       └── __pycache__
│           ├── __init__.cpython-310.pyc
│           └── __init__.cpython-39.pyc
│       └── utils
│           ├── __init__.py
│           ├── labmenu.py
│           ├── psycopg_types.py
│           ├── __pycache__
│           │   ├── __init__.cpython-310.pyc
│           │   ├── __init__.cpython-39.pyc
│           │   ├── labmenu.cpython-310.pyc
│           │   ├── labmenu.cpython-39.pyc
│           │   ├── psycopg_types.cpython-310.pyc
│           │   └── psycopg_types.cpython-39.pyc
│           └── View.py
│               ├── __init__.py
│               ├── __pycache__
│               │   ├── __init__.cpython-310.pyc
│               │   ├── __init__.cpython-39.pyc
│               │   ├── View.cpython-310.pyc
│               │   └── View.cpython-39.pyc
│               └── View.py
├── lab2.py3
├── postgres.ini.secure
├── *.py
└── README.md

10 directories, 44 files
lvizav@vizav-virtualbox lab2db1$
```

Рис. 3 – Структура програмного коду

Структура меню програми

Головне меню

```
MVC schema "DVD_rental_store" interface
  "DVD-rental" table
  "DVD-disk" table
  "loan" table
  "client" table
> Schema "DVD_rental_store" utils
  Dynamic search
  exit
```

Меню для таблиці

```
"DVD_rental_store"."DVD-rental" table interface:
> describe
  show data
  add data
  edit data
  remove data
  random fill
  return
```

Меню для вибору динамічних запитів

```
Schema "DVD_rental_store" dynamic search interface
> Disk
  Loan
  Client
  return
```

```
Disk dynamic search interface
"DiskName" ignored
"genre" ignored
"date" ignored
"price" ignored
"RentalName" ignored
"RentalAddress" ignored
"RentalOwner" ignored
> DiskName
  genre
  date
  price
  RentalName
  RentalAddress
  RentalOwner
  execute
  sql
  reset
  return
```

Завдання 1

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

Внесення даних

Створення нового тарифу:

```
id | address | name | owner
0 rows, execution time: 0:00:00.000911
"DVD_rental_store"."DVD-rental" table interface:
describe
> show data
add data
edit data
remove data
random fill
return
```

```
address: 1
name: ABC
owner: CBA
1 rows added
```

id	address	name	owner
1	1	ABC	CBA

Видалення даних

```
"DVD_rental_store"."DVD-rental" table interface:
describe
show data
add data
edit data
> remove data
random fill
return
```

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

id	address	name	owner
0 rows, execution time: 0:00:00.000281			

Якщо введено неіснуючий id рядку:

```
id: 2
0 rows deleted
```

Неправильно введене число:

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

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 | address | name | owner
2  | 1       | QWERTYasdfgh | ZXC
1 rows, execution time: 0:00:00.000394
"DVD_rental_store"."DVD-rental" table interface:
describe
show data
add data
> edit data
remove data
random fill
return
```

```
id: 2
address: 111
name: asdfgh
owner: qwe
1 rows changed
```

```
id | address | name | owner
2  | 111     | asdfgh | qwe
1 rows, execution time: 0:00:00.000317
```


Завдання 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 “Tariff”:

```
[vizav@vizav-virtualbox lab2db]$ ./lab2.py3
"DVD_rental_store"."DVD-rental" table interface:
describe
show data
add data
edit data
remove data
> random fill
return
```

```
instances [100]: 10
INSERT INTO "DVD_rental_store"."DVD-rental"("address", "name", "owner")
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)
FROM
    (VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters),
    generate_series(1, 10) as q;
```

"DVD_rental_store"."DVD-rental" 10 rows added, execution time: 0:00:00.005253

id	address	name	owner
1	nmQWERTYUI	QWERTYUIOP	VBNM
2	jklzxcvbnm	bnmQWERTYU	cvbnmQWERT
3	XCVBNM	klzxcvbnmQ	rtyuiopasd
4	OPASDFGHJK	YUIOPASDFG	bnmQWERTYU
5	bnmQWERTYU	opasdfghjk	TYUIOPASDF
6	tyuiopasdf	HJKLZXCVBN	cvbnmQWERT
7	CVBNM	fghjklzxcv	sdfghjklzx
8	YUIOPASDFG	KLZXCVBNM	SDFGHJKLZX
9	jklzxcvbnm	fghjklzxcv	BNM
10	yuioasdfg	CVBNM	SDFGHJKLZX

10 rows, execution time: 0:00:00.000410

"DVD_rental_store"."DVD-rental" table interface:

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

Пакетне заповнення таблиць схеми:

Кількість заданих рядків пакетного заповнення таблиць схеми:

DVD-rental	1_000
DVD-disk	1_000
client	2_000
loan	2_000
"DVD_rental_store"."DVD-rental" 1000 rows added, execution time: 0:00:00.017281	
"DVD_rental_store"."DVD-disk" 1000 rows added, execution time: 0:00:00.218152	
"DVD_rental_store"."client" 2000 rows added, execution time: 0:00:00.007473	
"DVD_rental_store"."loan" 2000 rows added, execution time: 0:00:01.003261	

DVD-rental

984	SDFGHJKLZX	cvbnmQWERT	BNM
985	vbnmQWERTY	klzxcvbnmQ	
986	DFGHJKLZXC	pasdfghjkl	asdfghjklz
987	QWERTYUIOP	KLZXCVBNM	IOPASDFGHJ
988	wertyuiopa	RTYUIOPASD	ZXCVBNM
989	cvbnmQWERT	asdfghjklz	XCVBNM
990	GHJKLZXCVB	pasdfghjkl	CVBNM
991	yuiopasdfg	PASDFGHJKL	M
992	qwertyuiop	QWERTYUIOP	WERTYUIOPA
993	HJKLZXCVBN	uiopasdfgh	ertyuiopas
994	jklzxcvbnm	pasdfghjkl	wertyuiopa
995	SDFGHJKLZX	SDFGHJKLZX	JKLZXCVBNM
996	ertyuiopas	LZXCVBNM	klzxcvbnmQ
997	HJKLZXCVBN	asdfghjklz	VBNM
998	bnmQWERTYU	KLZXCVBNM	WERTYUIOPA
999	dfghjklzxc	bnmQWERTYU	DFGHJKLZXC
1000	asdfghjklz	ERTYUIOPAS	ASDFGHJKLZ
1000 rows, execution time: 0:00:00.000985			

DVD-disk

984	293	nmQWERTYUI	dfghjklzxc	2021-03-23	09:53:42.853428+02:00	\$97.00
985	832		WERTYUIOPA	2021-09-22	08:57:02.816020+03:00	\$91.00
986	321	jklzxcvbnm	TYUIOPASDF	2021-11-09	08:14:06.868969+02:00	\$80.00
987	459	RTYUIOPASD	WERTYUIOPA	2021-03-10	22:37:13.326207+02:00	\$10.00
988	674	ghjklzxcvb	tyuiopasdf	2021-07-13	19:35:41.567885+03:00	\$22.00
989	529	ZXCVBNM	ERTYUIOPAS	2021-10-19	12:57:34.138885+03:00	\$99.00
990	568	tyuiopasdf	rtyuiopasd	2021-08-23	09:51:35.469033+03:00	\$25.00
991	500	nmQWERTYUI	tyuiopasdf	2021-06-24	11:33:03.476270+03:00	\$43.00
992	394	pasdfghjkl	ghjklzxcvb	2021-04-09	15:12:29.723285+03:00	\$83.00
993	689	JKLZXCVBNM	rtyuiopasd	2021-03-11	09:41:45.472869+02:00	\$91.00
994	574	HJKLZXCVBN	ertyuiopas	2021-04-28	21:38:33.677687+03:00	\$88.00
995	844	klzxcvbnmQ	QWERTYUIOP	2021-01-10	22:52:49.396257+02:00	\$50.00
996	336	rtyuiopasd	ERTYUIOPAS	2021-07-05	11:56:52.098544+03:00	\$91.00
997	884	lzcxcvbnmQW	ASDFGHJKLZ	2021-09-22	08:51:51.561937+03:00	\$86.00
998	196	rtyuiopasd	qwertyuiop	2021-01-03	13:55:03.311266+02:00	\$25.00
999	373	pasdfghjkl	FGHJKLZXC	2021-01-24	20:11:58.575106+02:00	\$62.00
1000	681	zxcvbnmQWE	IOPASDFGHJ	2021-07-23	16:44:36.264762+03:00	\$90.00

client

```
1984 | QWERTYUIOP | WERTYUIOPA
1985 | VBNM       | klzxcvbnmQ
1986 | WERTYUIOPA | FGHJKLZXC
1987 | GHJKLZXCVB | xcvbnmQWER
1988 | BNM        | dfghjklzxc
1989 | iopasdfghj | ertyuiopas
1990 | opasdfghjk | yuiopasdfg
1991 | ertyuiopas | jklzxcvbnm
1992 | HJKLZXCVCB | opasdfghjk
1993 | WERTYUIOPA | YUIOPASDFG
1994 | yuiopasdfg | NM
1995 | NM         | HJKLZXCVCB
1996 | HJKLZXCVCB | klzxcvbnmQ
1997 | QWERTYUIOP | M
1998 | qwertyuiop | ghjklzxcvb
1999 | ZXCVCBNM   | cvbnmQWERT
2000 | NM         | PASDFGHJKL
2000 rows, execution time: 0:00:00.001128
```

Loan

```
1984 | 705 | 1512 | 2021-08-21 11:24:26.783543+03:00 | 2021-07-25 10:16:44.472860+03:00 | klzxcvbnmQ
1985 | 329 | 812 | 2021-09-04 22:33:08.388415+03:00 | 2021-08-31 00:55:13.305493+03:00 | dfghjklzxc
1986 | 221 | 1895 | 2021-06-19 18:16:33.106531+03:00 | 2021-03-22 01:47:39.291404+02:00 | M
1987 | 494 | 572 | 2021-08-05 01:35:56.575036+03:00 | 2021-08-08 19:08:20.109067+03:00 | NM
1988 | 438 | 373 | 2021-02-13 18:21:46.213495+02:00 | 2021-10-11 05:39:40.378740+03:00 | tyuiopasdf
1989 | 958 | 1711 | 2021-06-12 12:22:38.641707+03:00 | 2021-07-06 21:03:48.285111+03:00 | pasdfghjkl
1990 | 996 | 1872 | 2021-08-19 09:17:16.167727+03:00 | 2021-10-13 18:00:27.031821+03:00 | VBNM
1991 | 539 | 859 | 2021-03-28 10:47:34.160766+03:00 | 2021-08-23 06:31:49.836219+03:00 | klzxcvbnmQ
1992 | 450 | 1043 | 2021-02-19 06:52:11.301911+02:00 | 2021-09-11 09:32:46.162794+03:00 | ERTYUIOPAS
1993 | 772 | 793 | 2021-06-10 15:58:08.860259+03:00 | 2021-08-14 10:08:22.158258+03:00 | WERTYUIOPA
1994 | 173 | 1824 | 2021-01-29 23:23:23.145272+02:00 | 2021-08-17 03:34:22.530340+03:00 | rtyuiopasd
1995 | 163 | 1802 | 2021-07-28 01:29:43.063749+03:00 | 2021-02-07 14:09:52.823429+02:00 | xcvbnmQWER
1996 | 316 | 1811 | 2021-01-13 04:35:13.617993+02:00 | 2021-08-20 19:37:32.534567+03:00 | DFGHJKLZXC
1997 | 641 | 1294 | 2021-06-16 05:50:14.409945+03:00 | 2021-05-07 04:52:18.201919+03:00 | BNM
1998 | 749 | 1572 | 2021-09-15 04:23:14.225061+03:00 | 2021-03-09 19:27:27.635953+02:00 | mQWERTYUIO
1999 | 703 | 40 | 2021-07-14 05:08:41.445983+03:00 | 2021-04-04 07:32:01.832124+03:00 | dfghjklzxc
2000 | 833 | 1503 | 2021-09-04 23:31:10.423586+03:00 | 2021-02-19 10:28:08.549954+02:00 | lzxcvbnmQW
2000 rows, execution time: 0:00:00.002587
```

SQL запити рандомізованого заповнення:

```
INSERT INTO
"DVD_rental_store"."DVD-rental"("address", "name", "owner")
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)
FROM
(VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVCBNM')) as symbols(characters),
generate_series(1, 1000) as q;

INSERT INTO
"DVD_rental_store"."DVD-disk"("DVD-rental_id", "name", "genre", "date", "price")
SELECT
(SELECT "id" FROM "DVD_rental_store"."DVD-rental" ORDER BY random()*q LIMIT 1),
substr(characters, (random() * length(characters) + 1)::integer, 10),
substr(characters, (random() * length(characters) + 1)::integer, 10),
timestamp '2021-01-01' + random() * (timestamp '2021-11-11' - timestamp '2021-01-01'),
trunc(random() * 100)::int
FROM
(VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVCBNM')) as symbols(characters),
generate_series(1, 1000) as q;
```



```

INSERT INTO
"DVD_rental_store"."client"("name", "surname")
SELECT
substr(characters, (random() * length(characters) + 1)::integer, 10),
substr(characters, (random() * length(characters) + 1)::integer, 10)
FROM
(VALUE('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters),
generate_series(1, 2000) as q;

INSERT INTO
"DVD_rental_store"."loan"("DVD-disk_id", "client_id", "date_loan", "date_return", "status")
SELECT
(SELECT "id" FROM "DVD_rental_store"."DVD-disk" ORDER BY random()*q LIMIT 1),
(SELECT "id" FROM "DVD_rental_store"."client" ORDER BY random()*q LIMIT 1),
timestamp '2021-01-01' + random() * (timestamp '2021-11-11' - timestamp '2021-01-01'),
timestamp '2021-01-01' + random() * (timestamp '2021-11-11' - timestamp '2021-01-01'),
substr(characters, (random() * length(characters) + 1)::integer, 10)
FROM
(VALUE('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters),
generate_series(1, 2000) as q;

```

Завдання 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 дисків за атрибутами:
 - назва диску
 - жанр
 - дата випуску
 - ціна
 - назва прокату
 - адреса прокату
 - власник прокату
- Пошук позик за атрибутами:
 - дата та час здійснення позики
 - дата та час повернення
 - статус
 - ім'я клієнта
 - прізвище клієнта
 - назва диску
 - жанр
 - дата випуску
 - ціна
- Пошук клієнта за атрибутами:
 - ім'я клієнта
 - прізвище клієнта
 - id клієнта

Пошук дисків:

SQL запит без фільтрації рядків:

```
SELECT
    "a"."name" as "DiskName",
    "a"."genre" as "genre",
    "a"."date" as "date",
    "a"."price" as "price",

    "b"."name" as "DiskName",
    "b"."address" as "RentalAddress",
    "b"."owner" as "RentalOwner"
FROM
    "DVD_rental_store"."DVD-disk" as "a"
```

```
INNER JOIN "DVD_rental_store"."DVD-rental" as "b"
ON "a"."DVD-rental_id" = "b"."id";
```

Результат:

```
DiskName | genre | date | price | DiskName | RentalAddress | RentalOwner
QWERTYUIOP | QWERTYUIOP | 2021-11-09 04:15:11.067191+02:00 | $21.00 | opasdfghjk | IOPASDFGHJ | UIOPASDFGH
1 rows, execution time: 0:00:00.004520
Disk dynamic search interface
"DiskName" ignored
"genre" LIKE 'Q%':varchar
"date" >= '2021-10-30 00:00:00':timestamp
"price" ignored
"RentalName" ignored
"RentalAddress" ignored
"RentalOwner" ignored
```

Налаштування фільтрування рядків:

- ім'я клієнта LIKE 'Q%'
- дата видачі >= 2021-10-30

SQL запит з заданими налаштуваннями фільтрування рядків:

```
SELECT
    "a"."name" as "DiskName",
    "a"."genre" as "genre",
    "a"."date" as "date",
    "a"."price" as "price",

    "b"."name" as "DiskName",
    "b"."address" as "RentalAddress",
    "b"."owner" as "RentalOwner"
FROM
    "DVD_rental_store"."DVD-disk" as "a"
    INNER JOIN "DVD_rental_store"."DVD-rental" as "b"
        ON "a"."DVD-rental_id" = "b"."id"
WHERE
    ("a"."genre" LIKE 'Q%':varchar) AND
    ("a"."date" >= '2021-10-30 00:00:00':timestamp);
```

Результат:

```
DiskName | genre | date | price | DiskName | RentalAddress | RentalOwner
QWERTYUIOP | QWERTYUIOP | 2021-11-09 04:15:11.067191+02:00 | $21.00 | opasdfghjk | IOPASDFGHJ | UIOPASDFGH
1 rows, execution time: 0:00:00.004520
Disk dynamic search interface
"DiskName" ignored
"genre" LIKE 'Q%':varchar
"date" >= '2021-10-30 00:00:00':timestamp
"price" ignored
"RentalName" ignored
"RentalAddress" ignored
"RentalOwner" ignored
```

Пошук позик:

SQL запит без фільтрації рядків:

```
SELECT
    "a"."date_loan" as "date_loan",
    "a"."date_return" as "date_return",
    "a"."status" as "status",

    "b"."name" as "ClientName",
    "b"."surname" as "ClientSurname",

    "c"."name" as "DiskName",
    "c"."genre" as "genre",
    "c"."date" as "date",
    "c"."price" as "price"

    -- "d"."name" as "DiskName",
```

```
--"d"."address" as "RentalAddress",
--"d"."owner" as "RentalOwner"
```

FROM

```
"DVD_rental_store"."loan" as "a"
INNER JOIN "DVD_rental_store"."client" as "b"
    ON "a"."client_id" = "b"."id"
INNER JOIN "DVD_rental_store"."DVD-disk" as "c"
    ON "a"."DVD-disk_id" = "c"."id"
INNER JOIN "DVD_rental_store"."DVD-rental" as "d"
    ON "c"."DVD-rental_id" = "d"."id";
```

Результат:

2021-07-27 10:58:24.357243+03:00	2021-07-26 10:16:09.410249+03:00	tyuiopasdf	JKLZXCVBNM	pasdfghjkl	pasdfghjkl	pasdfghjkl	2021-05-20 02:13:39.068854+03:00	\$37.00
2021-09-02 01:37:01.426859+03:00	2021-03-09 16:31:26.424259+02:00	ZXCVBNM	zxcvbnmQWE	vbnmQWERTY	RTYUIOPASD	YUIOPASDFG	2021-09-10 09:27:34.906593+03:00	\$74.00
2021-07-28 11:06:39.806747+03:00	2021-03-21 17:14:42.397180+02:00	VBNM	sdfghjklzxc	yuioasdfgh	SDFGHJKLZX	cvbnmQWERT	2021-07-29 12:15:00.933971+03:00	\$36.00
2021-08-15 00:56:52.578012+03:00	2021-02-09 23:42:54.512894+02:00	ERTYUIOPAS	mQWERTYUIO	RTYUIOPASD	QWERTYUIOP	cvbnmQWERT	2021-10-30 23:59:25.399902+03:00	\$32.00
2021-04-14 12:15:17.903311+03:00	2021-01-16 22:54:45.890073+03:00	ijklzxcvbnm	dghjklzxc	bnmQWERTYU	CVBNM	FGHJKLZXCV	2021-08-11 01:08:57.182854+03:00	\$23.00
2021-09-16 23:23:53.961262+03:00	2021-09-22 22:36:54.948669+03:00	ghjklzxcvb	uiopasdfgh	YUIOPASDFG	rtuyuiopasd	JKLZXCVBNM	2021-03-08 19:45:59.855784+02:00	\$61.00
2021-01-22 06:15:26.402462+02:00	2021-05-26 22:03:32.477849+03:00	ERTYUIOPAS	WERTYUIOPA	ZXCVBNM	LZXCVBNM	HJKLZXCVBN	2021-08-14 13:24:06.065885+03:00	\$22.00
2021-09-09 09:45:31.368398+03:00	2021-06-10 12:01:10.796949+03:00	yuioasdfgh	IOPASDFGHJ	BNM	pasdfghjkl	pasdfghjkl	2021-05-20 02:13:39.068854+03:00	\$37.00
2021-10-11 21:37:02.618636+03:00	2021-10-02 16:49:17.185315+03:00	pasdfghjkl	BNM	nmQWERTYUI	FGHJKLZXCV	SDFGHJKLZX	2021-08-13 01:17:02.229218+03:00	\$55.00
2021-02-07 04:11:49.041243+02:00	2021-07-12 02:30:44.849472+03:00	UIOPASDFGH	ijklzxcvbnmQ	cvbnmQWERT	ghjklzxcvb	ijklzxcvbnm	2021-09-22 08:32:12.392734+03:00	\$59.00
2021-03-15 22:32:56.275058+02:00	2021-05-20 04:19:46.178754+03:00	OPASDFGHJK	uiopasdfgh	ertyuiopas	vbnmQWERTY	DFGHJKLZX	2021-06-09 10:05:33.920995+03:00	\$67.00
2021-03-25 04:41:31.679774+02:00	2021-05-10 15:46:31.926030+03:00	ghjklzxcvb	BNM	YUIOPASDFG	FGHJKLZXCV	CVBNM	2021-08-20 19:44:27.487651+03:00	\$13.00
2021-10-28 03:28:23.870046+03:00	2021-03-24 10:58:46.038492+02:00	ASDFGHJKLZ	PASDFGHJKL	IOPASDFGHJ	SDFGHJKLZX	SDFGHJKLZX	2021-05-04 13:16:53.400310+03:00	\$48.00
2021-01-18 20:16:22.179386+02:00	2021-02-18 10:05:15.400524+02:00	ijklzxcvbnm	uiopasdfgh	opasdfghjk	opasdfghjk	opasdfghjk	2021-07-14 06:50:59.995300+03:00	\$99.00
2021-04-17 05:27:17.102597+03:00	2021-04-15 19:46:20.157115+03:00	NM	yuioasdfgh	IOPASDFGHJ	fgghjklzxcv	WERTYUIOPA	2021-04-03 06:47:34.232410+03:00	\$1.00
2021-06-10 01:52:49.514165+03:00	2021-05-05 05:44:38.431505+03:00	bnmQWERTYU	dghjklzxc	cvbnmQWERT	OPASDFGHJK	GHJKLZXCVB	2021-08-17 02:56:20.543439+03:00	\$45.00
2021-08-09 09:32:02.277774+03:00	2021-03-04 16:55:51.149237+02:00	PASDFGHJKL	RTYUIOPASD	ertyuiopas	IOPASDFGHJ	IOPASDFGHJ	2021-09-24 18:35:47.800259+03:00	\$49.00
2021-09-23 19:02:56.117165+03:00	2021-08-04 20:26:55.314787+03:00	PASDFGHJKL	WERTYUIOPA	QWERTYUIOP	dghjklzxc	M	2021-01-17 11:48:43.772889+02:00	\$49.00
2021-01-17 20:07:03.919541+02:00	2021-03-13 23:14:34.340979+02:00	RTYUIOPASD	rtuyuiopasd	uiopasdfgh	pasdfghjkl	VBNM	2021-10-02 03:00:19.214280+03:00	\$48.00
2021-09-19 02:05:19.653939+03:00	2021-06-18 20:11:22.355053+03:00	rtuyuiopasd	dghjklzxc	WERTYUIOPA	pasdfghjkl	asdfghjklz	2021-03-27 17:44:38.343973+02:00	\$79.00
2021-05-23 06:25:50.462388+03:00	2021-05-17 21:47:06.930647+03:00	fgghjklzxcv	nmQWERTYUI	lzxvbnmQW	FGHJKLZXCV	ijklzxcvbnm	2021-06-06 08:53:25.699436+03:00	\$94.00
2021-06-29 23:11:28.663579+03:00	2021-01-26 18:19:04.236113+02:00	hijklzxcvbn	WERTYUIOPA	PASDFGHJKL	vbnmQWERTY	QWERTYUIOP	2021-03-17 14:08:03.579475+02:00	\$82.00
2021-02-15 22:22:19.302560+02:00	2021-02-28 13:08:07.389594+02:00	opasdfghjk	HJKLZXCVBN	xcvbnmQWER	lzxvbnmQW	OPASDFGHJK	2021-03-17 03:56:31.117779+02:00	\$83.00
2021-09-13 18:01:22.841473+03:00	2021-06-17 02:52:29.829804+03:00	bnmQWERTYU	ioasdfghj	klzxcvbnmQ	SDFGHJKLZX	yuioasdfgh	2021-06-11 10:03:40.658315+03:00	\$15.00
2021-10-20 07:21:48.513331+03:00	2021-11-03 17:25:16.703186+02:00	XCVBNM	KLZXCVBNM	fgghjklzxcv	QWERTYUIOP	DFGHJKLZX	2021-09-17 21:57:47.138577+03:00	\$12.00
2021-04-06 12:56:01.451042+03:00	2021-11-10 06:47:07.661964+02:00	KLZXCVBNM	NM	WERTYUIOPA	VBNM	GHJKLZXCVB	2021-08-22 07:24:00.311362+03:00	\$58.00
2021-10-11 03:07:51.055969+03:00	2021-11-10 13:39:25.206371+02:00	IOPASDFGHJ	ijklzxcvbnmQ	lzxvbnmQW	HJKLZXCVBN	ijklzxcvbn	2021-03-15 17:48:06.214753+02:00	\$61.00
2021-04-24 16:15:50.030422+03:00	2021-01-27 08:22:04.708983+02:00	QWERTYUIOP	NM	JKLZXCVBNM	sdfghjklzxc	JKLZXCVBNM	2021-09-27 12:41:18.578430+03:00	\$57.00
2021-10-19 22:57:02.326094+03:00	2021-03-06 01:03:28.696570+02:00	XCVBNM	RTYUIOPASD	ERTYUIOPAS	BNM	ertyuiopas	2021-05-28 07:00:42.242040+03:00	\$71.00
2021-07-29 07:46:56.602504+03:00	2021-08-19 15:51:22.853684+03:00	YUIOPASDFG	ijklzxcvbnmQ	GHJKLZXCVB	ERTYUIOPAS	JKLZXCVBNM	2021-04-11 11:25:01.417156+03:00	\$81.00
2021-02-22 03:21:30.101159+02:00	2021-06-27 11:43:27.673177+03:00	cvbnmQWERT	VBNM	UIOPASDFGH	lzxvbnmQW	CVBNM	2021-04-25 02:25:15.415265+03:00	\$16.00
2021-02-07 17:11:42.350331+02:00	2021-02-07 19:48:06.317300+02:00	RTYUIOPASD	WERTYUIOPA	ZXCVBNM	pasdfghjklz	pasdfghjklz	2021-04-10 07:37:55.085859+03:00	\$5.00
2021-06-05 23:12:27.192352+03:00	2021-06-26 00:56:44.596264+03:00	qwertuiopas	sdfghjklzxc	hijklzxcvbn	fgghjklzxcv	rtuyuiopasd	2021-03-23 06:13:00.407212+02:00	\$39.00
2021-07-13 09:03:30.787139+03:00	2021-07-30 13:32:58.486981+03:00	UIOPASDFGH	ioasdfghj	ijklzxcvbnm	SDFGHJKLZX	UIOPASDFGH	2021-01-15 14:52:37.444722+02:00	\$46.00
2021-06-27 05:30:29.125740+03:00	2021-03-09 16:35:03.273754+02:00	qwertuiopas	uiopasdfgh	vbnmQWERTY	JKLZXCVBNM	M	2021-07-07 03:54:32.574722+03:00	\$73.00
2021-04-25 13:06:43.653091+03:00	2021-08-27 08:27:18.379853+03:00	KLZXCVBNM	uiopasdfgh	QWERTYUIOP	IOPASDFGHJ	nmQWERTYUIO	2021-03-15 04:17:30.041808+02:00	\$61.00
2021-01-05 20:55:58.280347+02:00	2021-07-01 12:50:21.440559+03:00	UIOPASDFGH	OPASDFGHJK	XCVBNM	FGHJKLZXCV	uiopasdfgh	2021-07-13 04:45:17.707134+03:00	\$81.00

2000 rows, execution time: 0:00:00.007416

loan dynamic search interface

"date_loan" ignored

"date_return" ignored

"status" ignored

"ClientName" ignored

"ClientSurname" ignored

"DiskName" ignored

"genre" ignored

"date" ignored

"price" ignored

Налаштування фільтрування рядків:

- дата видачі <= 2021-09-01
- назва диску LIKE 'i%'
- ціна < 25\$

SQL запит з заданими налаштуваннями фільтрування рядків:

SELECT

```
"a"."date_loan" as "date_loan",
"a"."date_return" as "date_return",
"a"."status" as "status",
```

```
"b"."name" as "ClientName",
"b"."surname" as "ClientSurname",
```

```
"c"."name" as "DiskName",
"c"."genre" as "genre",
"c"."date" as "date",
"c"."price" as "price"
```

```
--"d"."name" as "DiskName",
--"d"."address" as "RentalAddress",
--"d"."owner" as "RentalOwner"
```

FROM

```
"DVD_rental_store"."loan" as "a"
INNER JOIN "DVD_rental_store"."client" as "b"
```

```

ON "a"."client_id" = "b"."id"
INNER JOIN "DVD_rental_store"."DVD-disk" as "c"
ON "a"."DVD-disk_id" = "c"."id"
INNER JOIN "DVD_rental_store"."DVD-rental" as "d"
ON "c"."DVD-rental_id" = "d"."id"

WHERE
("a"."date_loan" <= '2021-09-01 00:00:00'::timestamp) AND
("c"."name" LIKE 'i% '::varchar) AND
("c"."price" < 25.0::money);

```

Результат:

date_loan	date_return	status	ClientName	ClientSurname	DiskName	genre	date	price
2021-03-21 00:16:07.453938+02:00	2021-02-13 06:27:11.891184+02:00	SDFGHJKLZX	mOWERTYUIO	asdfghjklz	iopasdfghj	dfghjklzxc	2021-07-10 13:10:31.985495+03:00	\$21.00
2021-01-14 12:54:00.079204+02:00	2021-07-17 01:52:32.456622+03:00	lzxcvbnmQW	NN	uiopasdfgh	iopasdfghj	FGHJKLZXCV	2021-08-19 06:02:43.186767+03:00	\$15.00
2021-03-22 17:54:28.138982+02:00	2021-07-13 03:31:45.597581+03:00	XCVBNM	JKLZXCVBNM	sdfghjklz	iopasdfghj	YUIOPASDFG	2021-02-11 09:51:56.043339+02:00	\$15.00
2021-04-23 09:21:52.963080+03:00	2021-09-28 16:12:12.110048+03:00	WERTYUIOPA	zxcvbnmQWE	vbnmQWERTY	iopasdfghj	YUIOPASDFG	2021-02-11 09:51:56.043339+02:00	\$15.00
2021-08-01 02:03:29.893253+03:00	2021-05-03 22:57:24.403493+03:00	jklzxcvbnm	ghjklzxcvb	lzxcvbnmQW	iopasdfghj	GHJKLZXCVB	2021-07-06 09:15:29.682591+03:00	\$0.00
2021-03-30 03:27:50.361293+03:00	2021-04-22 15:56:27.576455+03:00	asdfghjklz	yuioasdfgh	IOPASDFGHJ	iopasdfghj	sdfghjklz	2021-10-12 17:43:53.149765+03:00	\$20.00
2021-03-15 17:50:29.386150+02:00	2021-06-26 21:35:54.407898+03:00	zxcvbnmQWE	xcvbnmQWER	vbnmQWERTY	iopasdfghj	TVUIOPASDF	2021-02-28 10:31:26.954331+02:00	\$11.00
2021-04-19 00:45:38.724987+03:00	2021-06-29 07:34:23.291938+03:00	BNM	opasdfghjk	ZXCVBNM	iopasdfghj	sdfghjklz	2021-10-12 17:43:53.149765+03:00	\$20.00
2021-04-15 19:20:26.569071+03:00	2021-05-07 15:30:48.610464+03:00	DFGHJKLZX	opasdfghjk	UIOPASDFGH	iopasdfghj	dfghjklzxc	2021-07-10 13:10:31.985495+03:00	\$21.00

9 rows, execution time: 0:00:00.001633

Loan dynamic search interface
 "date_loan" <= '2021-09-01 00:00:00'::timestamp
 "date_return" ignored
 "status" ignored
 "ClientName" ignored
 "ClientSurname" ignored
 "DiskName" LIKE 'i% '::varchar
 "genre" ignored
 "date" ignored
 "price" < 25.0::money

Пошук клієнтів:

SQL запит без фільтрації рядків:

```

SELECT
  "a"."id" as "id",
  "a"."name" as "name",
  "a"."surname" as "surname"
FROM
  "DVD_rental_store"."client" as "a";

```

Результат:

1987	JKLZXCVBNM	ASDFGHJKLZ
1988	RTYUIOPASD	uiopasdfgh
1989	lzxcvbnmQW	pasdfghjkl
1990	SDFGHJKLZX	SDFGHJKLZX
1991	JKLZXCVBNM	pasdfghjkl
1992	IOPASDFGHJ	PASDFGHJKL
1993	GHJKLZXCVB	pasdfghjkl
1994	rtyuiopasd	hijklzxcvbn
1995	OPASDFGHJK	rtyuiopasd
1996	lzxcvbnmQW	BNM
1997	XCVBNM	tyuiopasdf
1998	VBNM	iopasdfghj
1999	SDFGHJKLZX	hijklzxcvbn
2000	rtyuiopasd	ghijklzxcvb

2000 rows, execution time: 0:00:00.001286

Налаштування фільтрування рядків:

- id клієнта >= 150
- id клієнта < 170

SQL запит з заданими налаштуваннями фільтрування рядків:

```

SELECT
  "a"."id" as "id",
  "a"."name" as "name",
  "a"."surname" as "surname"
FROM
  "DVD_rental_store"."client" as "a"

```


WHERE

("a"."id" >= 150::bigint AND "a"."id" < 170::bigint);

Результат:

150	VBNM	tyuiopasdf
151	pasdfghjkl	VBNM
152	iopasdfghj	YUIOPASDFG
153	zxcvbnmQWE	ERTYUIOPAS
154	pasdfghjkl	opasdfghjk
155	dfghjklzxc	ertyuiopas
156	iopasdfghj	CVBNM
157	asdfghjklz	WERTYUIOPA
158	KLZXCVBNM	sdfghjklzx
159	hijklzxcvbn	wertyuiopa
160	CVBNM	ghijklzxcvb
161	IOPASDFGHJ	klzxcvbnmQ
162	VBNM	dfghjklzxc
163	QWERTYUIOP	cvbnmQWERT
164	sdfghjklzx	yuiopasdfg
165	xcvbnmQWER	mQWERTYUIO
166	FGHJKLZXC	bnmQWERTYU
167		ertyuiopas
168	NM	yuiopasdfg
169	CVBNM	opasdfghjk

20 rows, execution time: 0:00:00.000518

Client dynamic search interface

"id" >= 150::bigint AND < 170::bigint

"name" ignored

"surname" ignored

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.psycopg2_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.constraint_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',
references_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
dbcursor.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.table}';
    """
    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, (psycopg2.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.primary_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.columns() 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_replace:
str = None):

        if sql_replace:
            pass
        else:
            if instances is None:
                return Lab.utils.menuInput(self.randomFill,
[collections.namedtuple("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.references_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.references_field is
            None else psql_foreign_key_random(x), columns))}
        FROM

        (VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) 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._tables.keys())})"

    def __getattr__(self, name):
        try:
            if name in ["_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"str {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
optimized
    #     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()

```

```
#!/usr/bin/env python
import itertools
import pprint

from .dynamicsearch import *

__all__ = [f"DiskDynamicSearch", f"LoanDynamicSearch", f"ClientDynamicSearch", ]

class DiskDynamicSearch(DynamicSearchBase):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.name: str = "Disk"
        self.search: dict[self.SearchCriterias[CompareConstant]] = {
            "DiskName": SearchCriterias(f'"a"."name"', f"name", f"varchar"),
            "genre": SearchCriterias(f'"a"."genre"', f"genre", f"varchar"),
            "date": SearchCriterias(f'"a"."date"', f"date", f"timestamp"),
            "price": SearchCriterias(f'"a"."price"', f"price", f"money"),

            "RentalName": SearchCriterias(f'"b"."name"', f"name",
f"varchar"),
            "RentalAddress": SearchCriterias(f'"b"."address"', f"address",
f"varchar"),
            "RentalOwner": SearchCriterias(f'"b"."owner"', f"owner",
f"varchar"),
        }

    @property
    def sql(self):
        where = self.where
        sql = f"""
        SELECT
            "a"."name" as "DiskName",
            "a"."genre" as "genre",
            "a"."date" as "date",
            "a"."price" as "price",

            "b"."name" as "DiskName",
            "b"."address" as "RentalAddress",
            "b"."owner" as "RentalOwner"
        FROM
            "{self.schema}"."DVD-disk" as "a"
            INNER JOIN "{self.schema}"."DVD-rental" as "b"
                ON "a"."DVD-rental_id" = "b"."id"
        {f' 'WHERE
            {where};' ' if where else f";"}
        """

        return sql

class LoanDynamicSearch(DynamicSearchBase):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.name: str = "Loan"
        self.search: dict[self.SearchCriterias[CompareConstant]] = {
```

```

        "date_loan": SearchCriterias(f'"a"."date_loan"', f"date_loan",
f"timestamp"),
        "date_return": SearchCriterias(f'"a"."date_return"',
f"date_return", f"timestamp"),
        "status": SearchCriterias(f'"a"."owner"', f"status", f"varchar"),

        "ClientName": SearchCriterias(f'"b"."name"', f"name",
f"varchar"),
        "ClientSurname": SearchCriterias(f'"b"."surname"', f"surname",
f"varchar"),

        "DiskName": SearchCriterias(f'"c"."name"', f"name", f"varchar"),
        "genre": SearchCriterias(f'"c"."genre"', f"genre", f"varchar"),
        "date": SearchCriterias(f'"c"."date"', f"date", f"timestamp"),
        "price": SearchCriterias(f'"c"."price"', f"price", f"money"),

        # "RentalName": SearchCriterias(f'"d"."name"', f"name",
f"varchar"),
        # "RentalAddress": SearchCriterias(f'"d"."address"', f"address",
f"varchar"),
        # "RentalOwner": SearchCriterias(f'"d"."owner"', f"owner",
f"varchar"),
    }

```

```

@property
def sql(self):
    where = self.where
    sql = f"""
        SELECT
            "a"."date_loan" as "date_loan",
            "a"."date_return" as "date_return",
            "a"."status" as "status",

            "b"."name" as "ClientName",
            "b"."surname" as "ClientSurname",

            "c"."name" as "DiskName",
            "c"."genre" as "genre",
            "c"."date" as "date",
            "c"."price" as "price"

            --"d"."name" as "DiskName",
            --"d"."address" as "RentalAddress",
            --"d"."owner" as "RentalOwner"

        FROM
            "{self.schema}"."loan" as "a"
            INNER JOIN "{self.schema}"."client" as "b"
                ON "a"."client_id" = "b"."id"
            INNER JOIN "{self.schema}"."DVD-disk" as "c"
                ON "a"."DVD-disk_id" = "c"."id"
            INNER JOIN "{self.schema}"."DVD-rental" as "d"
                ON "c"."DVD-rental_id" = "d"."id"
        {f''WHERE
            {where};'' if where else f'';}
    """

```

```

        return sql

class ClientDynamicSearch(DynamicSearchBase):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.name: str = "Client"
        self.search: dict[self.SearchCriterias[CompareConstant]] = {
            "id": SearchCriterias(f'"a"."id"', f"id", f"bigint"),
            "name": SearchCriterias(f'"a"."name"', f"name", f"varchar"),
            "surname": SearchCriterias(f'"a"."surname"', f"surname",
f"varchar"),
        }

    @property
    def sql(self):
        where = self.where
        sql = f"""
            SELECT
                "a"."id" as "id",
                "a"."name" as "name",
                "a"."surname" as "surname"
            FROM
                "{self.schema}"."client" as "a"
            {f'WHERE'
             {where};' if where else f'';}
        """

        return sql

    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")CompareConstant(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""""{" AND ".join(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""""{" AND ".join(f"{a}" for a in self if not
a.isIgnored)}""""
        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.promt = f"{self}"
#     return result

```

```

class SelectCompositor(object):
    def __init__(self, search_criterias, table):
        super().__init__()
        self._search_criterias: SearchCriterias[CompareConstant] =
search_criterias
        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 enumerate(self.search_criterias, 1)},
                # "new criteria": lambda:
self.search_criterias[self.table].append(),
                "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
```

```
from . import DynamicSearch
from .AutoSchema import *
```

```
class DVD_rental_store(Schema):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self._dynamicsearch = {a.name: a for a in
[DynamicSearch.DiskDynamicSearch(self), DynamicSearch.LoanDynamicSearch(self),
DynamicSearch.ClientDynamicSearch(self), ]}
        # self.reoverride()

    def reoverride(self):
        # Table override
        # self._tables.DVDrental = SchemaTable(self, f"DVD-rental")
        # self._tables.DVDdisk = SchemaTable(self, f"DVDdisk")
        # self._tables.client = SchemaTable(self, f"client")
        # self._tables.loan = SchemaTable(self, f"loan")
        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}."DVD-rental" (
                "id" bigserial PRIMARY KEY,
                "address" character varying(255) NOT NULL,
                "name" character varying(255) NOT NULL,
                "owner" character varying(255) NOT NULL
            );
            """,
            f"""CREATE TABLE "{self}."DVD-disk" (
                "id" bigserial PRIMARY KEY,
                "DVD-rental_id" bigint NOT NULL,
                "name" character varying(255) NOT NULL,
                "genre" character varying(255) NOT NULL,
                "date" timestamp with time zone NOT NULL,
                "price" money NOT NULL,

                CONSTRAINT "DVD-disk_DVD-rental_id_fkey" FOREIGN KEY
("DVD-rental_id")
                REFERENCES "{self}."DVD-rental("id") MATCH SIMPLE
                ON UPDATE NO ACTION
                ON DELETE CASCADE
```

```

        NOT VALID
    );
    """
    f"""CREATE TABLE "{self}."client" (
        "id" bigserial PRIMARY KEY,
        "name" character varying(255) NOT NULL,
        "surname" character varying(255) NOT NULL
    );
    """
    f"""CREATE TABLE "{self}."loan" (
        "id" bigserial PRIMARY KEY,
        "DVD-disk_id" bigint NOT NULL,
        "client_id" bigint NOT NULL,
        "date_loan" timestamp with time zone NOT NULL,
        "date_return" timestamp with time zone NOT NULL,
        "status" character varying(255) NOT NULL,
        CONSTRAINT "loan_DVD-disk_id_fkey" FOREIGN KEY ("DVD-
disk_id")
        REFERENCES "{self}."DVD-disk"("id") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE
        NOT VALID,
        CONSTRAINT "loan_client_id_fkey" FOREIGN KEY ("client_id")
        REFERENCES "{self}."client"("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):
    getattr(self.tables, f"DVD-rental").randomFill(1_000)
    getattr(self.tables, f"DVD-disk").randomFill(1_000)
    self.tables.client.randomFill(2_000)
    self.tables.loan.randomFill(2_000)

def _test():
    pass

if __name__ == "__main__":
    _test()

__init__.py

```

```

from .Schema import *

__all__ = ["DVD_rental_store"]

```

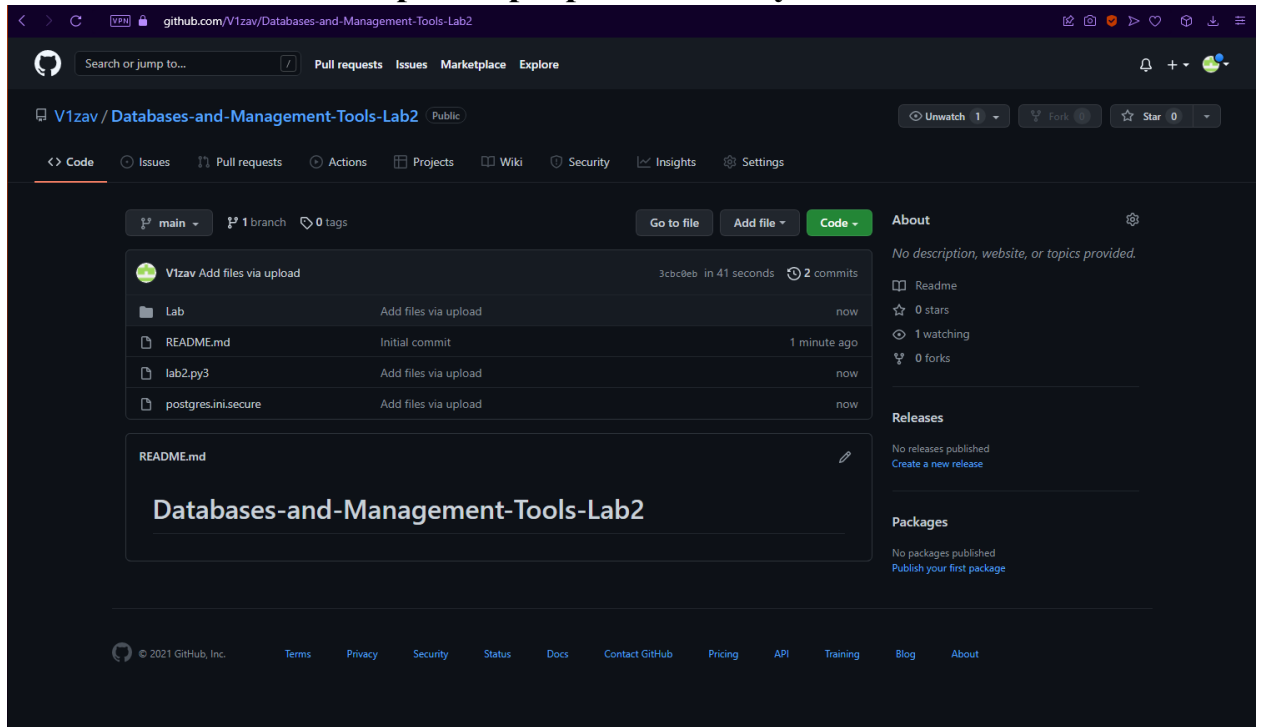
Короткий опис функцій

Файл Schema.py складається з класу DVD_rental_store.

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



Посилання на репозиторій: <https://github.com/V1zav/Databases-and-Management-Tools-Lab2>