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
Микитенко І. П.
Перевірив: 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).

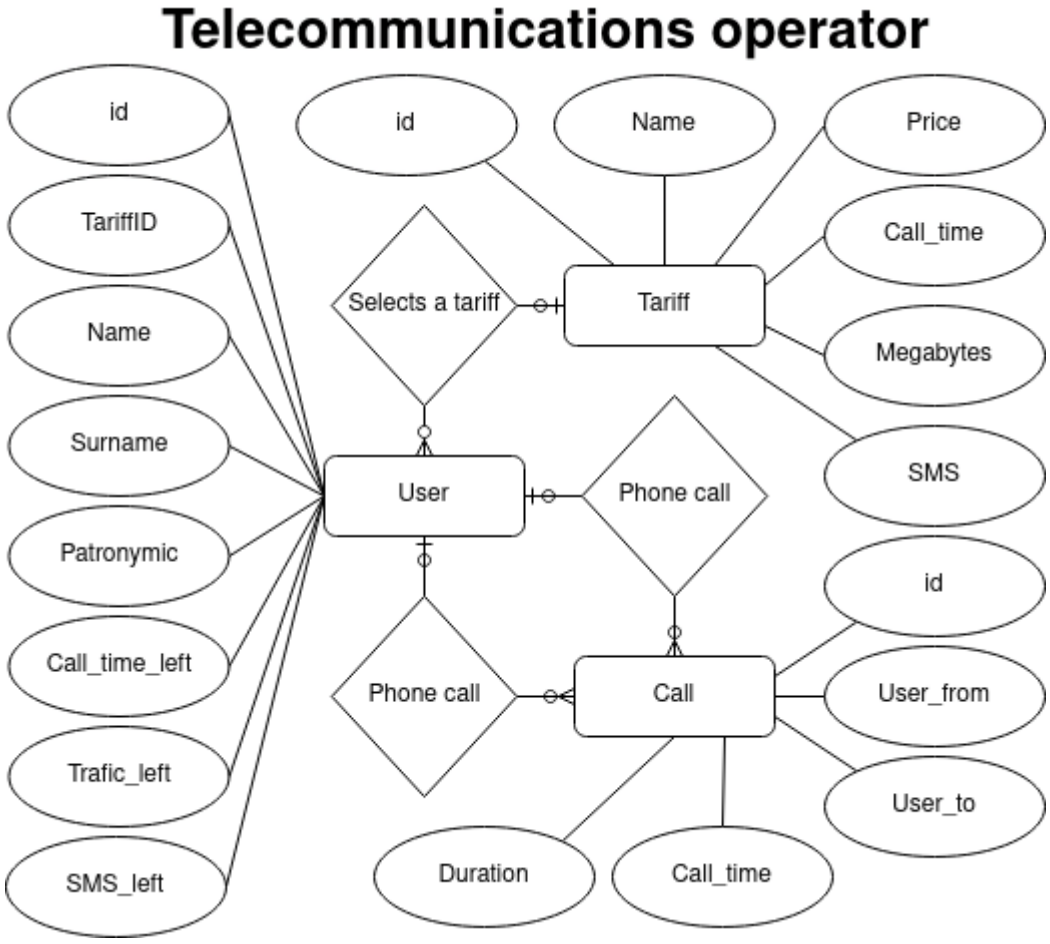
Деталізоване завдання:

1. Забезпечити можливість уведення/редагування/вилучення даних у таблицях бази даних з можливістю контролю відповідність 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 підлеглій 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.

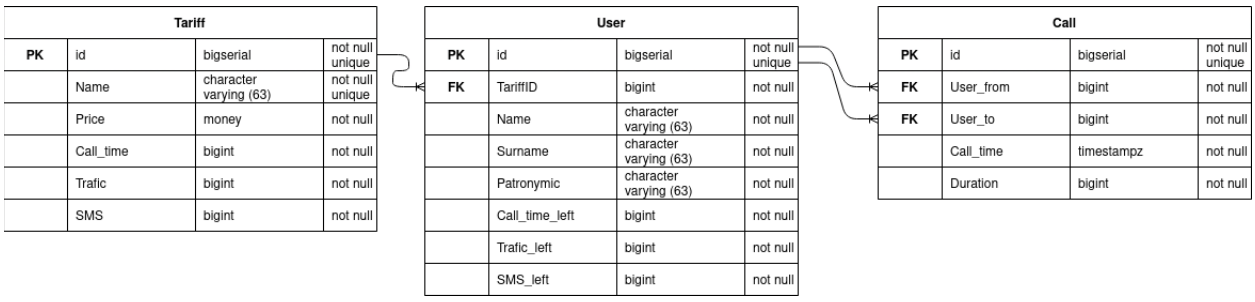
4. Програмний код організувати згідно шаблону Model-View-Controller(MVC). При цьому модель, подання та контролер мають бути реалізовані у окремих файлах. Для доступу до бази даних використовувати **лише мову SQL** (без ORM).

Інформація про модель та структуру бази даних

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



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



Tariff

PK	id	bigserial	not null unique
	Name	character varying (63)	not null unique
	Price	money	not null
	Call_time	bigint	not null
	Trafic	bigint	not null
	SMS	bigint	not null

User

PK	id	bigserial	not null unique
FK	TariffID	bigint	not null
	Name	character varying (63)	not null
	Surname	character varying (63)	not null
	Patronymic	character varying (63)	not null
	Call_time_left	bigint	not null
	Trafic_left	bigint	not null
	SMS_left	bigint	not null

Call

PK	id	bigserial	not null unique
FK	User_from	bigint	not null
FK	User_to	bigint	not null
	Call_time	timestampz	not null
	Duration	bigint	not null

Рис. 2 – Логічна модель бази даних

Зміни у порівнянні з першою лабораторною роботою відсутні.

Середовище розробки та налаштування підключення до бази даних

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

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

Опис структури програми

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

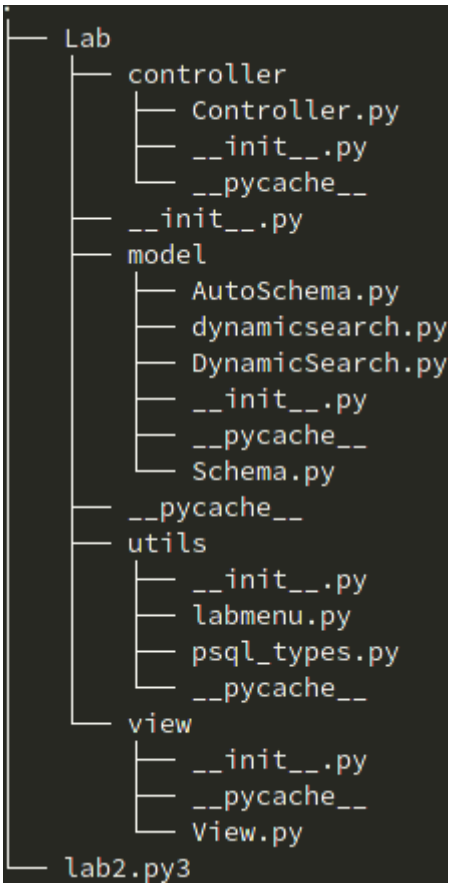


Рис. 3 – Структура програмного коду

Структура меню програми

Головне меню

```
MVC schema "Telecommunications" interface
> "Tariff" table
  "User" table
  "Call" table
  Schema "Telecommunications" utils
  Dynamic search
  exit
```

Меню для таблиці

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

Меню для вибору динамічних запитів

```
Schema "Telecommunications" dynamic search interface
> User
  Call
  return
```

```
User dynamic search interface
"Name" ignored
"Surname" ignored
"Patronymic" ignored
"Call_time_left" ignored
"Trafic_left" ignored
"SMS_left" ignored
"TariffName" ignored
"TariffPrice" ignored
"TariffCall_time" ignored
"TariffTrafic" ignored
"TariffSMS" ignored
> Name
  Surname
  Patronymic
  Call_time_left
  Trafic_left
  SMS_left
  TariffName
  TariffPrice
  TariffCall_time
  TariffTrafic
  TariffSMS
  execute
  sql
  reset
  return
```

Меню вибору кількості рядків для генерації

```
instances [100]:
```

Пункт 1

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

Внесення даних

Створення нового тарифу:

```
id  | Name  | Price  | Call_time  | Trafic  | SMS
0 rows, execution time: 0:00:00.000617
```

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

```
Name: XXXL
Price: 10000
Call_time: 999999999999999
Trafic: 99999999999999999
SMS: 99999999999999
1 rows added
```

```
id  | Name  | Price       | Call_time        | Trafic              | SMS
1   | XXXL  | $10,000.00  | 999999999999999  | 99999999999999999   | 99999999999999
1 rows, execution time: 0:00:00.000371
```

Видалення даних

```
id  | Name  | Price       | Call_time        | Trafic              | SMS
1   | XXXL  | $10,000.00  | 999999999999999  | 99999999999999999   | 99999999999999
1 rows, execution time: 0:00:00.000371
```

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

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

```
id  | Name  | Price  | Call_time  | Trafic  | SMS
0 rows, execution time: 0:00:00.000439
```

Якщо уведено неіснуючий id рядку:

```
id: 100
0 rows deleted
```

Неправильно введене число:

```
id: 500a
Error: 500a is not a valid integer
id:
```

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

Редагування даних

```
id  | TariffID  | Name      | Surname    | Patronymic  | Call_time_left  | Trafic_left  | SMS_left
1   | 1         | LZXCVBNM  | qwertyuiop | klzxcvbnmQ  | 52              | 20           | 94
1 rows, execution time: 0:00:00.000727
"Telecommunications"."User" table interface:
  describe
  show data
  add data
> edit data
  remove data
  random fill
  return
```

```
id: 1
TariffID: 1
Name: NEWNAME
Surname: NEWSURNAME
Patronymic: NEWPATRONYMIC
Call_time_left: 500a
Error: 500a is not a valid integer
Call_time_left: 800
Trafic_left: 543
SMS_left: 999
1 rows changed
```

```
id  | TariffID  | Name     | Surname     | Patronymic     | Call_time_left  | Trafic_left  | SMS_left
1   | 1         | NEWNAME  | NEWSURNAME  | NEWPATRONYMIC  | 800             | 543          | 999
1 rows, execution time: 0:00:00.000397
```

Пункт 2

Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.

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

Рандомізоване заповнення таблиці “Tariff”:

```
id  | Name  | Price  | Call_time  | Trafic  | SMS
0 rows, execution time: 0:00:00.000738
"Telecommunications"."Tariff" table interface:
  describe
  show data
  add data
  edit data
  remove data
> random fill
  return
```

```
instances [100]: 4
"Telecommunications"."Tariff" 4 rows added, execution time: 0:00:00.001311
```

```
id  | Name         | Price   | Call_time  | Trafic  | SMS
1   | M            | $99.00  | 96         | 12      | 46
2   | nmQWERTYUI   | $14.00  | 5          | 58      | 65
3   | ZXCVBNM      | $50.00  | 28         | 2       | 77
4   | RTYUIOPASD   | $71.00  | 50         | 74      | 13
4 rows, execution time: 0:00:00.000541
```

Пакетне заповнення таблиць схеми:

Кількість заданих рядків пакетного заповнення таблиць схеми:

Tariff	1_000
User	2_000
Call	10_000

```
"Telecommunications"."Tariff" 1000 rows added, execution time: 0:00:00.005861
"Telecommunications"."User" 2000 rows added, execution time: 0:00:00.541322
"Telecommunications"."Call" 10000 rows added, execution time: 0:00:13.737136
```

Витяги деяких рандомізованих рядків з таблиць:

Tariff

id	Name	Price	Call_time	Trafic	SMS
1	PASDFGHJKL	$52.00	37	33	33
2	ERTYUIOPAS	$65.00	91	44	10
3	LZXCVBNM	$97.00	0	24	52
4	fghjklzxcv	$3.00	47	60	67
5	WERTYUIOPA	$77.00	99	65	19
6	vbnmQWERTY	$10.00	2	53	83
7	wertyuiopa	$6.00	27	78	35
8	xcvbnmQWER	$28.00	66	0	78
9	ertyuiopas	$19.00	65	72	77
10	cvbnmQWERT	$97.00	78	66	21
11	ERTYUIOPAS	$63.00	36	66	62
12	asdfghjklz	$30.00	97	44	32
13	klzxcvbnmQ	$30.00	69	33	62
14	nmQWERTYUI	$57.00	35	4	52
15	FGHJKLZXCV	$8.00	68	73	38
16	JKLZXCVBNM	$83.00	96	53	41
17	YUIOPASDFG	$48.00	79	53	26
18	BNM	$22.00	5	59	95
19	tyuiopasdf	$95.00	48	0	48
20	ASDFGHJKLZ	$34.00	57	23	61

20 rows, execution time: 0:00:00.000538

User

id	TariffID	Name	Surname	Patronymic	Call_time_left	Trafic_left	SMS_left
1	11	asdfghjklz	nmQWERTYUI	VBNM	77	35	18
2	1	rtyuiopasd	cvbnmQWERT	ghjklzxcvb	49	39	62
3	18	SDFGHJKLZX	M	ertyuiopas	78	9	58
4	3		opasdfghjk	HJKLZXCVBN	79	68	50
5	6	fghjklzxcv	nmQWERTYUI	asdfghjklz	39	41	7
6	20	uiopasdfgh	NM	lzxcvbnmQW	61	84	85
7	3	ZXCVBNM	FGHJKLZXCV	asdfghjklz	29	0	0
8	5	dfghjklzxc	jklzxcvbnm	iopasdfghj	72	21	82
9	5	HJKLZXCVBN	SDFGHJKLZX	GHJKLZXCVB	84	39	85
10	18	sdfghjklzx	ZXCVBNM	ERTYUIOPAS	38	92	18
11	3	TYUIOPASDF	WERTYUIOPA	tyuiopasdf	1	20	22
12	7	YUIOPASDFG	bnmQWERTYU	JKLZXCVBNM	34	8	35
13	19	OPASDFGHJK	KLZXCVBNM	RTYUIOPASD	13	96	5
14	3	ERTYUIOPAS		HJKLZXCVBN	81	93	8
15	14	tyuiopasdf	pasdfghjkl	DFGHJKLZXC	79	53	47

15 rows, execution time: 0:00:00.000381

Call

id	User_from	User_to	Call_time	Duration
1	7	6	2021-03-25 13:19:48.710126+02:00	69
2	14	10	2021-07-18 00:57:07.354843+03:00	40
3	14	2	2021-06-26 18:43:49.768270+03:00	52
4	8	2	2021-10-19 01:46:11.008952+03:00	42
5	12	4	2021-03-03 15:30:35.716426+02:00	39
6	15	15	2021-03-15 19:46:39.165910+02:00	80
7	11	14	2021-08-01 08:46:23.322991+03:00	43
8	5	7	2021-05-05 22:59:47.898314+03:00	39
9	13	13	2021-07-07 09:06:36.402264+03:00	19
10	9	10	2021-10-29 07:44:17.199660+03:00	51
11	10	8	2021-05-16 21:12:19.564438+03:00	3
12	12	1	2021-01-28 17:12:44.734342+02:00	0
13	11	13	2021-07-04 20:26:51.965608+03:00	88
14	2	13	2021-08-24 04:08:19.587654+03:00	62
15	13	13	2021-08-07 13:32:21.057765+03:00	19
16	4	13	2021-02-09 17:21:16.015786+02:00	58
17	5	11	2021-02-20 15:06:16.338801+02:00	15
18	8	9	2021-04-04 16:25:27.195828+03:00	42
19	1	11	2021-10-30 12:18:03.102503+03:00	16
20	10	5	2021-02-19 10:14:58.975769+02:00	6

20 rows, execution time: 0:00:00.000751

SQL запити рандомізованого заповнення:

```
INSERT INTO "Telecommunications"."Call"("User_from", "User_to", "Call_time", "Duration")
                                SELECT(SELECT "id" FROM "Telecommunications"."User" ORDER BY
random()*q LIMIT 1),
                                (SELECT "id" FROM "Telecommunications"."User" ORDER BY random()*q
LIMIT 1),
                                timestamp '2021-01-01' + random() * (timestamp '2021-11-11' -
timestamp '2021-01-01'),
                                trunc(random() * 100)::int
                                FROM

(VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters),
                                        generate_series(1, 100) as q;
```

```
INSERT INTO "Telecommunications"."Tariff"("Name", "Price", "Call_time", "Trafic", "SMS")
                                SELECT
                                        substr(characters, (random() * length(characters) +
1)::integer, 10),
                                        trunc(random() * 100)::int,
                                        trunc(random() * 100)::int,
                                        trunc(random() * 100)::int,
                                        trunc(random() * 100)::int
                                FROM

(VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters),
                                        generate_series(1, 100) as q;
```

```
INSERT INTO "Telecommunications"."User"("TariffID", "Name", "Surname", "Patronymic",
"Call_time_left", "Trafic_left", "SMS_left")
                                SELECT
                                        (SELECT "id" FROM "Telecommunications"."Tariff" ORDER BY
random()*q LIMIT 1),
                                        substr(characters, (random() * length(characters) +
1)::integer, 10),
                                        substr(characters, (random() * length(characters) +
1)::integer, 10),
                                        substr(characters, (random() * length(characters) +
1)::integer, 10),
                                        trunc(random() * 100)::int,
                                        trunc(random() * 100)::int,
                                        trunc(random() * 100)::int
                                FROM

(VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters),
                                        generate_series(1, 100) as q;
```

Пункт 3

Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.

Було підготовлено два SQL запити:

- Пошук користувачів за атрибутами:
  - ім’я користувача
  - прізвище користувача
  - по батькові користувача
  - кількість доступних хвилин
  - кількість доступного трафіку
  - кількість доступних SMS
  - назва тарифу
  - ціна тарифу
  - кількість доступних хвилин за тарифом
  - кількість доступного трафіку за тарифом
  - кількість доступних SMS за тарифом
- Пошук дзвінків за атрибутами:
  - дата та час початку дзвінка
  - тривалість дзвінка
  - ім’я користувача що дзвонив
  - прізвище користувача що дзвонив
  - по батькові користувача що дзвонив
  - ім’я користувача якому дзвонили
  - прізвище користувача якому дзвонили

- по батькові користувача якому дзвонили

Пошук користувачів:
SQL запит без фільтрації рядків:

```
SELECT
        "a"."Call_time" as "Call_time",
        "a"."Duration" as "Duration",

        "b"."Name" as "FromName",
        "b"."Surname" as "FromSurname",
        "b"."Patronymic" as "FromPatronymic",

        "c"."Name" as "ToName",
        "c"."Surname" as "ToSurname",
        "c"."Patronymic" as "ToPatronymic"

FROM
        "Telecommunications"."Call" as "a"
        INNER JOIN "Telecommunications"."User" as "b"
                ON "a"."User_from" = "b"."id"
        INNER JOIN "Telecommunications"."User" as "c"
                ON "a"."User_to" = "c"."id"
;
```

Результат:

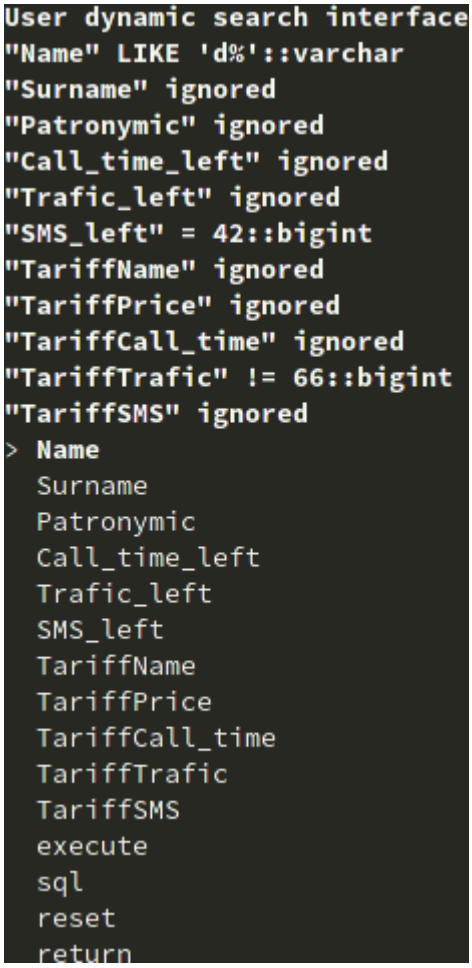
Name	Surname	Patronymic	Call_time_left	Trafic_left	SMS_left	TariffName	TariffPrice	TariffCall_time	Tarifffrafic	TariffSMS
rtyuiopasd	cvbnmQWERT	ghjklzxcvb	49	39	62	PASDFGHJKL	$52.00	37	33	33
ERTYUIOPAS		HJKLZXCVBN	81	93	8	LZXCVBNM	$97.00	0	24	52
TYUIOPASDF	WERTYUIOPA	tyuiopasdf	1	20	22	LZXCVBNM	$97.00	0	24	52
ZXCVBNM	FGHJKLZXCV	asdfghjklz	29	0	0	LZXCVBNM	$97.00	0	24	52
	opasdfghjk	HJKLZXCVBN	79	68	50	LZXCVBNM	$97.00	0	24	52
HJKLZXCVBN	SDFGHJKLZX	GHJKLZXCVB	84	39	85	WERTYUIOPA	$77.00	99	65	19
dfghjklzxc	jklzxcvbnm	iopasdfghj	72	21	82	WERTYUIOPA	$77.00	99	65	19
fghjklzxcv	nmQWERTYUI	asdfghjklz	39	41	7	vbnmQWERTY	$10.00	2	53	83
YUIOPASDFG	bnmQWERTYU	JKLZXCVBNM	34	8	35	wertyuiopa	$6.00	27	78	35
asdfghjklz	nmQWERTYUI	VBNM	77	35	18	ERTYUIOPAS	$63.00	36	66	62
tyuiopasdf	pasdfghjkl	DFGHJKLZXC	79	53	47	nmQWERTYUI	$57.00	35	4	52
sdfghjklzx	ZXCVBNM	ERTYUIOPAS	38	92	18	BNM	$22.00	5	59	95
SDFGHJKLZX	M	ertyuiopas	78	9	58	BNM	$22.00	5	59	95
OPASDFGHJK	KLZXCVBNM	RTYUIOPASD	13	96	5	tyuiopasdf	$95.00	48	0	48
uiopasdfgh	NM	lzxcvbnmQW	61	84	85	ASDFGHJKLZ	$34.00	57	23	61

15 rows, execution time: 0:00:00.001228

Налаштування фільтрування рядків:

- ім’я користувача LIKE ‘b%’
- кількість доступних SMS = 42
- кількість доступного трафіку за тарифом != 66

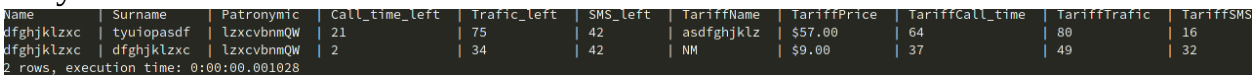
Введені критерії фільтрування:



SQL запит з заданими налаштуваннями фільтрування рядків:

```
SELECT
        "a"."Name" as "Name",
        "a"."Surname" as "Surname",
        "a"."Patronymic" as "Patronymic",
        "a"."Call_time_left" as "Call_time_left",
        "a"."Trafic_left" as "Trafic_left",
        "a"."SMS_left" as "SMS_left",
        "b"."Name" as "TariffName",
        "b"."Price" as "TariffPrice",
        "b"."Call_time" as "TariffCall_time",
        "b"."Trafic" as "TariffTrafic",
        "b"."SMS" as "TariffSMS"
FROM
        "Telecommunications"."User" as "a"
        INNER JOIN "Telecommunications"."Tariff" as "b"
                ON "a"."TariffID" = "b"."id"
WHERE
        ("a"."Name" LIKE 'd%'::varchar) AND
        ("a"."SMS_left" = 42::bigint) AND
        ("b"."Trafic" != 66::bigint);
```

Результат:



Name	Surname	Patronymic	Call_time_left	Trafic_left	SMS_left	TariffName	TariffPrice	TariffCall_time	TariffTrafic	TariffSMS
dfghjklzxc	tyuiopasdf	lzxcvbnmQW	21	75	42	asdfghjklz	$57.00	64	80	16
dfghjklzxc	dfghjklzxc	lzxcvbnmQW	2	34	42	NM	$9.00	37	49	32

2 rows, execution time: 0:00:00.001028

Пошук дзвінків:
SQL запит без фільтрації рядків:

```
SELECT
        "a"."Call_time" as "Call_time",
        "a"."Duration" as "Duration",

        "b"."Name" as "FromName",
        "b"."Surname" as "FromSurname",
        "b"."Patronymic" as "FromPatronymic",

        "c"."Name" as "ToName",
        "c"."Surname" as "ToSurname",
        "c"."Patronymic" as "ToPatronymic"

FROM
        "Telecommunications"."Call" as "a"
        INNER JOIN "Telecommunications"."User" as "b"
                ON "a"."User_from" = "b"."id"
        INNER JOIN "Telecommunications"."User" as "c"
                ON "a"."User_to" = "c"."id"
;
```

Результат:

```
2021-08-16 10:33:39.433780+03:00 | 49 | vbnmQWERTY | GHJKLZXCVB | tyuiopasdf | ghjklzxcvb | IOPASDFGHJ | zxcvbnmQWE
2021-02-11 22:23:16.653313+02:00 | 26 | bnmQWERTYU | fghjklzxcv | nmQWERTYUI | bnmQWERTYU | asdfghjklz | zxcvbnmQWE
2021-05-06 10:22:56.364024+03:00 | 2 | BNM | cvbnmQWERT | cvbnmQWERT | yuiopasdfg | mQWERTYUIO | FGHJKLZXCV
2021-09-25 03:59:55.792852+03:00 | 36 | LZXCVBNM | opasdfghjk | GHJKLZXCVB | TYUIOPASDF | bnmQWERTYU | HJKLZXCVBN
2021-10-17 22:04:13.152498+03:00 | 97 | PASDFGHJKL | SDFGHJKLZX | lzxcvbnmQW | vbnmQWERTY | klzxcvbnmQ | jklzxcvbnm
2021-01-23 00:59:48.785394+02:00 | 64 | tyuiopasdf | IOPASDFGHJ | bnmQWERTYU | QWERTYUIOP | OPASDFGHJK | xcvbnmQWER
2021-03-16 11:19:41.511296+02:00 | 48 | ghjklzxcvb | ZXCVBNM | ASDFGHJKLZ | opasdfghjk | mQWERTYUIO | rtyuiopasd
2021-05-01 02:37:11.164440+03:00 | 33 | ERTYUIOPAS | fghjklzxcv | xcvbnmQWER | xcvbnmQWER | ASDFGHJKLZ | cvbnmQWERT
2021-09-02 12:13:10.624797+03:00 | 45 | opasdfghjk | hjklzxcvbn | yuiopasdfg | iopasdfghj | wertyuiopa | NM
2021-07-23 13:03:12.373220+03:00 | 3 | uiopasdfgh | WERTYUIOPA | zxcvbnmQWE | GHJKLZXCVB | iopasdfghj | yuiopasdfg
2021-05-28 05:20:12.434920+03:00 | 69 | xcvbnmQWER | XCVBNM | FGHJKLZXCV | OPASDFGHJK | pasdfghjkl | vbnmQWERTY
2021-09-30 20:18:45.291122+03:00 | 99 | cvbnmQWERT | yuiopasdfg | ertyuiopas | UIOPASDFGH | NM | sdfghjklzx
2021-03-04 23:57:29.633812+02:00 | 79 | ertyuiopas | ZXCVBNM | wertyuiopa | xcvbnmQWER | ZXCVBNM | klzxcvbnmQ
2021-02-03 00:10:30.574326+02:00 | 51 | NM | xcvbnmQWER | CVBNM | ghjklzxcvb | KLZXCVBNM | XCVBNM
2021-02-21 21:48:23.725703+02:00 | 65 | | fghjklzxcv | zxcvbnmQWE | SDFGHJKLZX | cvbnmQWERT | vbnmQWERTY
2021-10-08 08:52:24.849643+03:00 | 64 | RTYUIOPASD | TYUIOPASDF | CVBNM | ASDFGHJKLZ | M | RTYUIOPASD
2021-01-30 17:57:02.995290+02:00 | 33 | bnmQWERTYU | hjklzxcvbn | klzxcvbnmQ | bnmQWERTYU | VBNM | CVBNM
2021-04-16 00:37:18.612084+03:00 | 26 | asdfghjklz | ghjklzxcvb | iopasdfghj | zxcvbnmQWE | qwertyuiop | ASDFGHJKLZ
2021-02-19 13:57:47.304942+02:00 | 30 | NM | IOPASDFGHJ | YUIOPASDFG | CVBNM | xcvbnmQWER | rtyuiopasd
2021-02-13 00:39:10.472592+02:00 | 13 | CVBNM | wertyuiopa | tyuiopasdf | pasdfghjkl | WERTYUIOPA | yuiopasdfg
2021-04-20 00:44:52.087365+03:00 | 21 | fghjklzxcv | GHJKLZXCVB | UIOPASDFGH | jklzxcvbnm | sdfghjklzx | mQWERTYUIO
2021-08-15 13:44:12.932612+03:00 | 1 | FGHJKLZXCV | XCVBNM | opasdfghjk | rtyuiopasd | BNM | pasdfghjkl
2021-06-08 04:22:01.454078+03:00 | 49 | XCVBNM | OPASDFGHJK | nmQWERTYUI | BNM | QWERTYUIOP | asdfghjklz
2021-04-28 22:23:04.876874+03:00 | 77 | DFGHJKLZXC | rtyuiopasd | RTYUIOPASD | iopasdfghj | iopasdfghj | sdfghjklzx
2021-10-29 14:18:23.753137+03:00 | 6 | ERTYUIOPAS | cvbnmQWERT | QWERTYUIOP | tyuiopasdf | lzxcvbnmQW | ertyuiopas
2021-08-02 14:05:41.224726+03:00 | 68 | KLZXCVBNM | ghjklzxcvb | iopasdfghj | DFGHJKLZXC | ZXCVBNM | nmQWERTYUI
2021-03-02 08:18:35.931996+02:00 | 44 | QWERTYUIOP | ghjklzxcvb | klzxcvbnmQ | ZXCVBNM | iopasdfghj | ertyuiopas
2021-10-08 22:48:44.413813+03:00 | 64 | opasdfghjk | rtyuiopasd | CVBNM | YUIOPASDFG | opasdfghjk | mQWERTYUIO
10000 rows, execution time: 0:00:00.017969
Call dynamic search interface
"Call_time" ignored
"Duration" ignored
"FromName" ignored
"FromSurname" ignored
"FromPatronymic" ignored
"ToName" ignored
"ToSurname" ignored
"ToPatronymic" ignored
> Call_time
  Duration
  FromName
  FromSurname
  FromPatronymic
  ToName
  ToSurname
  ToPatronymic
  execute
  sql
  reset
  return
```

Налаштування фільтрування рядків:

- дата та час початку дзвінка >= 2021-05-13 00:00:00
- дата та час початку дзвінка <= 2021-07-14 00:00:00
- тривалість дзвінка >= 22
- тривалість дзвінка <= 50
- ім’я користувача що дзвонив LIKE 'V%'

SQL запит з заданими налаштуваннями фільтрування рядків:

```
SELECT
        "a"."Call_time" as "Call_time",
        "a"."Duration" as "Duration",

        "b"."Name" as "FromName",
        "b"."Surname" as "FromSurname",
        "b"."Patronymic" as "FromPatronymic",

        "c"."Name" as "ToName",
        "c"."Surname" as "ToSurname",
        "c"."Patronymic" as "ToPatronymic"

        FROM
                "Telecommunications"."Call" as "a"
                INNER JOIN "Telecommunications"."User" as "b"
                        ON "a"."User_from" = "b"."id"
                INNER JOIN "Telecommunications"."User" as "c"
                        ON "a"."User_to" = "c"."id"
        WHERE
                ("a"."Call_time" >= '2021-05-13 00:00:00'::timestamp AND
"a"."Call_time" < '2021-07-14 00:00:00'::timestamp) AND
                ("a"."Duration" >= 35::bigint AND "a"."Duration" <=
40::bigint) AND
                ("b"."Name" LIKE 'V%'::varchar);
```

Результат:

```
Call_time                       | Duration | FromName | FromSurname | FromPatronymic | ToName      | ToSurname   | ToPatronymic
2021-05-21 21:58:21.144981+03:00 | 39       | VBNM     | KLZXCVBNM   | fghjklzxcv     | BNM         | OPASDFGHJK  | zxcvbnmQWE
2021-06-30 16:21:45.679747+03:00 | 36       | VBNM     | JKLZXCVBNM  |                | lzxcvbnmQW  | hjklzxcvbn  | bnmQWERTYU
2021-06-28 09:55:49.038374+03:00 | 37       | VBNM     | pasdfghjkl  | NM             | VBNM        | vbnmQWERTY  | JKLZXCVBNM
2021-05-27 03:15:37.213898+03:00 | 38       | VBNM     | XCVBNM      | SDFGHJKLZX     | GHJKLZXCVB  | cvbnmQWERT  | nmQWERTYUI
2021-07-07 11:31:02.878891+03:00 | 38       | VBNM     | ERTYUIOPAS  | NM             | opasdfghjk  | cvbnmQWERT  | lzxcvbnmQW
2021-06-18 03:01:04.154970+03:00 | 40       | VBNM     | cvbnmQWERT  | mQWERTYUIO     | PASDFGHJKL  | TYUIOPASDF  | tyuiopasdf
6 rows, execution time: 0:00:00.004075
Call dynamic search interface
"Call_time" >= '2021-05-13 00:00:00'::timestamp AND < '2021-07-14 00:00:00'::timestamp
"Duration" >= 35::bigint AND <= 40::bigint
"FromName" LIKE 'V%'::varchar
"FromSurname" ignored
"FromPatronymic" ignored
"ToName" ignored
"ToSurname" ignored
"ToPatronymic" ignored
> Call_time
  Duration
  FromName
  FromSurname
  FromPatronymic
  ToName
  ToSurname
  ToPatronymic
  execute
  sql
  reset
```

Код програмного модулю model

AutoSchema.py

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

import re
import Lab.utils
import collections

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

import Lab.utils.psql_types


__all__ = ["SchemaTable", "Schema"]


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

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

                self.schema = schema
                self.table = table

                self.primary_key_name = f"id"

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

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

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

```
                sql = f"""
                        SELECT
                                tb.table_schema, tb.table_name,
tb.column_name, tb.data_type, tb.is_nullable,
                                fx.constraint_name, fx.references_schema,
fx.references_table, fx.references_field
                        FROM information_schema.columns tb
                        LEFT JOIN (
                                SELECT
                                        tc.constraint_schema,
                                        tc.table_name,
                                        kcu.column_name,
                                        tc.constraint_name,
                                        tc.constraint_type,
                                        rc.update_rule AS on_update,
                                        rc.delete_rule AS on_delete,
                                        ccu.constraint_schema AS
references_schema,
                                        ccu.table_name AS references_table,
                                        ccu.column_name AS references_field
                                FROM information_schema.table_constraints tc
                                LEFT JOIN information_schema.key_column_usage
kcu
                                        ON tc.constraint_catalog =
kcu.constraint_catalog
                                        AND tc.constraint_schema =
kcu.constraint_schema
                                        AND tc.constraint_name =
kcu.constraint_name
                                LEFT JOIN
information_schema.referential_constraints rc
                                        ON tc.constraint_catalog =
rc.constraint_catalog
                                        AND tc.constraint_schema =
rc.constraint_schema
                                        AND tc.constraint_name =
rc.constraint_name
                                LEFT JOIN
information_schema.constraint_column_usage ccu
                                        ON rc.unique_constraint_catalog =
ccu.constraint_catalog
                                        AND rc.unique_constraint_schema =
ccu.constraint_schema
                                        AND rc.unique_constraint_name =
ccu.constraint_name
                                WHERE tc.constraint_schema NOT ILIKE 'pg_%'
AND tc.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 promt(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"),
```

```
                }, promt=self.promt)
                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 [f"_tables"]:
                                raise KeyError
                        return self._tables[name]
                except KeyError as e:
                        try:
                                return super().__getattribute__(name)
                        except KeyError as e:
                                raise AttributeError(f"{name} is not known
table")

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

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

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

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

        # def __next__(self):
                # try:
                #       result = tuple(self._tables.values())[self._iter]  #
may be 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(sef):
                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 promt(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"),
                                }, promt=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"),
                                }, promt=f"""Schema "{self}" dynamic search
interface""")
                                #self.dynamicsearch,
                }, promt=self.promt)

                return result


def _test():
        pass
```

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

DynamicSearch.py

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

from .dynamicsearch import *

__all__ = ["UserDynamicSearch", "CallDynamicSearch"]


class UserDynamicSearch(DynamicSearchBase):
        def __init__(self, *args, **kwargs):
                super().__init__(*args, **kwargs)
                self.name: str = "User"
                self.search: dict[self.SearchCriterias[CompareConstant]] = {
                        "Name": SearchCriterias(f'"a"."Name"', f"Name",
"varchar"),
                        "Surname": SearchCriterias(f'"a"."Surname"',
f"Surname", "varchar"),
                        "Patronymic": SearchCriterias(f'"a"."Patronymic"',
"Patronymic", "varchar"),

                        "Call_time_left":
SearchCriterias(f'"a"."Call_time_left"', f"Call_time_left", "bigint"),
                        "Trafic_left": SearchCriterias(f'"a"."Trafic_left"',
f"Trafic_left", "bigint"),
                        "SMS_left": SearchCriterias(f'"a"."SMS_left"',
f"SMS_left", "bigint"),

                        "TariffName": SearchCriterias(f'"b"."Name"',
f"TariffName", "varchar"),
                        "TariffPrice": SearchCriterias(f'"b"."Price"',
f"TariffPrice", "money"),
                        "TariffCall_time":
SearchCriterias(f'"b"."Call_time"', f"TariffCall_time", "bigint"),
                        "TariffTrafic": SearchCriterias(f'"b"."Trafic"',
f"TariffTrafic", "bigint"),
                        "TariffSMS": SearchCriterias(f'"b"."SMS"',
f"TariffSMS", "bigint"),
                }

        @property
        def sql(self):
                where = self.where
                sql = f"""
                        SELECT
                                "a"."Name" as "Name",
                                "a"."Surname" as "Surname",
                                "a"."Patronymic" as "Patronymic",

                                "a"."Call_time_left" as "Call_time_left",
                                "a"."Trafic_left" as "Trafic_left",
                                "a"."SMS_left" as "SMS_left",
```

```
                        "b"."Name" as "TariffName",
                        "b"."Price" as "TariffPrice",
                        "b"."Call_time" as "TariffCall_time",
                        "b"."Trafic" as "TariffTrafic",
                        "b"."SMS" as "TariffSMS"

                FROM
                        "{self.schema}"."User" as "a"
                        INNER JOIN "{self.schema}"."Tariff" as "b"
                                ON "a"."TariffID" = "b"."id"
                {f'''WHERE
                        {where};''' if where else f";"}
        """

        return sql

class CallDynamicSearch(DynamicSearchBase):
        def __init__(self, *args, **kwargs):
                super().__init__(*args, **kwargs)
                self.name: str = "Call"
                self.search: dict[self.SearchCriterias[CompareConstant]] = {
                        "Call_time": SearchCriterias(f'"a"."Call_time"',
"Call_time", "timestamp"),
                        "Duration": SearchCriterias(f'"a"."Duration"',
"Duration", "bigint"),

                        "FromName": SearchCriterias(f'"b"."Name"',
f"FromName", "varchar"),
                        "FromSurname": SearchCriterias(f'"b"."Surname"',
f"FromSurname", "varchar"),
                        "FromPatronymic":
SearchCriterias(f'"b"."Patronymic"', "FromPatronymic", "varchar"),

                        "ToName": SearchCriterias(f'"c"."Name"', f"ToName",
"varchar"),
                        "ToSurname": SearchCriterias(f'"c"."Surname"',
f"ToSurname", "varchar"),
                        "ToPatronymic": SearchCriterias(f'"c"."Patronymic"',
"ToPatronymic", "varchar"),
                }

        @property
        def sql(self):
                where = self.where
                sql = f"""
                        SELECT
                                "a"."Call_time" as "Call_time",
                                "a"."Duration" as "Duration",

                                "b"."Name" as "FromName",
                                "b"."Surname" as "FromSurname",
                                "b"."Patronymic" as "FromPatronymic",

                                "c"."Name" as "ToName",
                                "c"."Surname" as "ToSurname",
                                "c"."Patronymic" as "ToPatronymic"

                        FROM
```

```
                                "{self.schema}"."Call" as "a"
                                INNER JOIN "{self.schema}"."User" as "b"
                                        ON "a"."User_from" = "b"."id"
                                INNER JOIN "{self.schema}"."User" as "c"
                                        ON "a"."User_to" = "c"."id"
                        {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 promt(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"),
                }, promt=self.promt)
                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 promt(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"),
                        }, promt=self.promt)
                        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 promt(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"),
                        }, promt=self.promt)
                        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 Telecommunications(Schema):
        def __init__(self, *args, **kwargs):
                super().__init__(*args, **kwargs)
                self._dynamicsearch = {a.name: a for a in
[DynamicSearch.UserDynamicSearch(self),
DynamicSearch.CallDynamicSearch(self)]}
                # self.reoverride()

        def reoverride(self):
                # Table override
                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}"."Tariff" (
                                "id" bigserial PRIMARY KEY,
                                "Name" character varying(63) NOT NULL,
                                "Price" money NOT NULL, --
                                "Call_time" bigint NOT NULL, -- milliseconds
                                "Trafic" bigint NOT NULL, -- bytes
                                "SMS" bigint NOT NULL
                                -- UNIQUE("Name")
                        );
                        """,
                        f"""CREATE TABLE "{self}"."User" (
                                "id" bigserial PRIMARY KEY,
                                "TariffID" bigint NOT NULL,
                                "Name" character varying(63) NOT NULL,
                                "Surname" character varying(63) NOT NULL,
                                "Patronymic" character varying(63) NOT NULL,
                                "Call_time_left" bigint NOT NULL, --
milliseconds
                                "Trafic_left" bigint NOT NULL, -- bytes
                                "SMS_left" bigint NOT NULL,
                                CONSTRAINT "User_TariffID_fkey" FOREIGN KEY
("TariffID")
                                        REFERENCES
"Telecommunications"."Tariff"("id") MATCH SIMPLE
                                        ON UPDATE NO ACTION
                                        ON DELETE CASCADE
                                        NOT VALID
                        );
                        """,
                        f"""CREATE TABLE "{self}"."Call" (
                                "id" bigserial PRIMARY KEY,
                                "User_from" bigint NOT NULL,
                                "User_to" bigint NOT NULL,
                                "Call_time" timestamp with time zone NOT
NULL,
                                "Duration" bigint NOT NULL DEFAULT 0, --
milleseconds
                                CONSTRAINT "Call_User_from_fkey" FOREIGN KEY
("User_from")
                                        REFERENCES
"Telecommunications"."User"("id") MATCH SIMPLE
                                        ON UPDATE NO ACTION
                                        ON DELETE CASCADE
                                        NOT VALID,
```

```
                                CONSTRAINT "Call_User_to_fkey" FOREIGN KEY
("User_to")
                                        REFERENCES
"Telecommunications"."User"("id") MATCH SIMPLE
                                        ON UPDATE NO ACTION
                                        ON DELETE CASCADE
                                        NOT VALID
                        );
                        """,
                ]

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

                self.dbconn.commit()

                self.refresh_tables()

        def randomFill(self):
                self.tables.Tariff.randomFill(1_000)
                self.tables.User.randomFill(2_000)
                self.tables.Call.randomFill(10_000)


def _test():
        pass


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

__init__.py

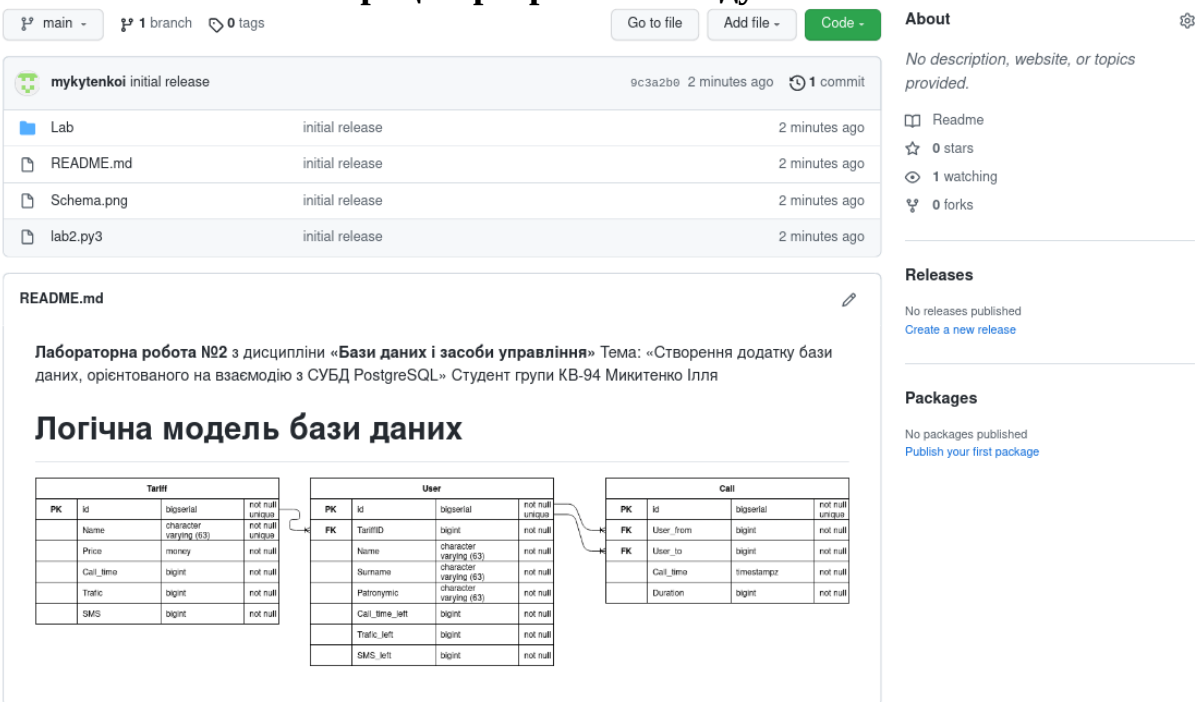
```
from .Schema import *

__all__ = ["Telecommunications"]
```

Короткий опис функцій

Файл Schema.py складається з класу Telecommunications.
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/mykytenkoi/Databases-and-Management-Tools-Lab2