НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ім. Ігоря СІКОРСЬКОГО»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ
Кафедра системного програмування і спеціалізованих комп'ютерних систем

Лабораторна робота №2

з дисципліни

«Бази даних і засоби управління»

*Тема**: «Проектування бази даних та ознайомлення з базовими операціями СУБД PostgreSQL»***

Виконав: студент III курсу
ФПМ групи КВ-94
Колесніков Є. О.
Перевірив: Петрашенко А.В.

Київ — 2021

Завдання

Загальне завдання роботи полягає у наступному:

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

URL репозиторію

GitHub: ***https://github.com/kolesnikov-dev/DB_lab2***

Інструментарій

Мова програмування: Python.
Використані бібліотеки: psycopg2, time

Діаграма сутність-зв’язок

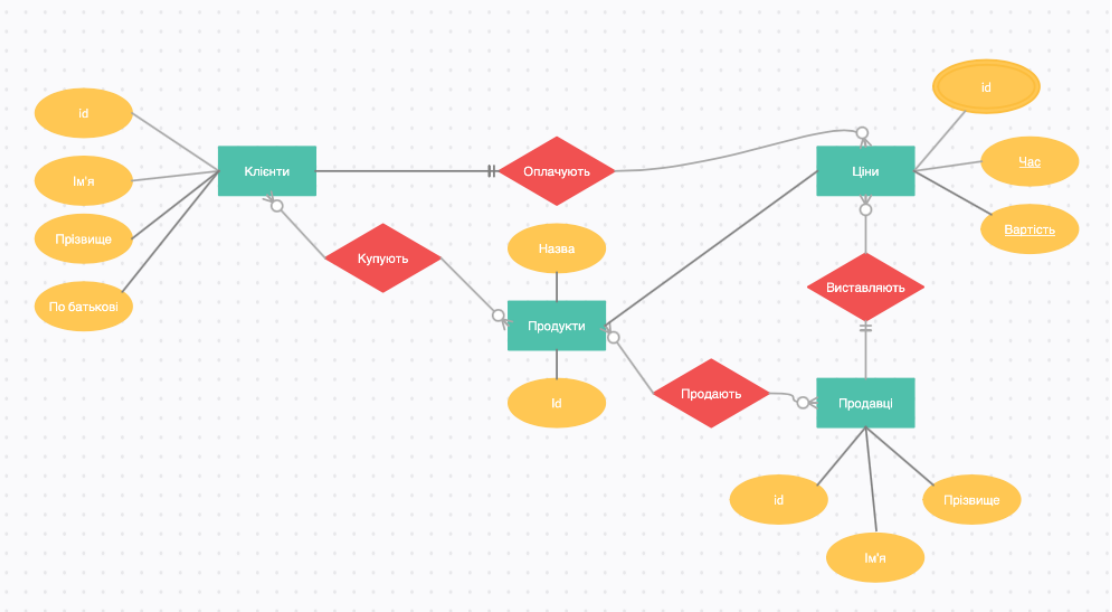
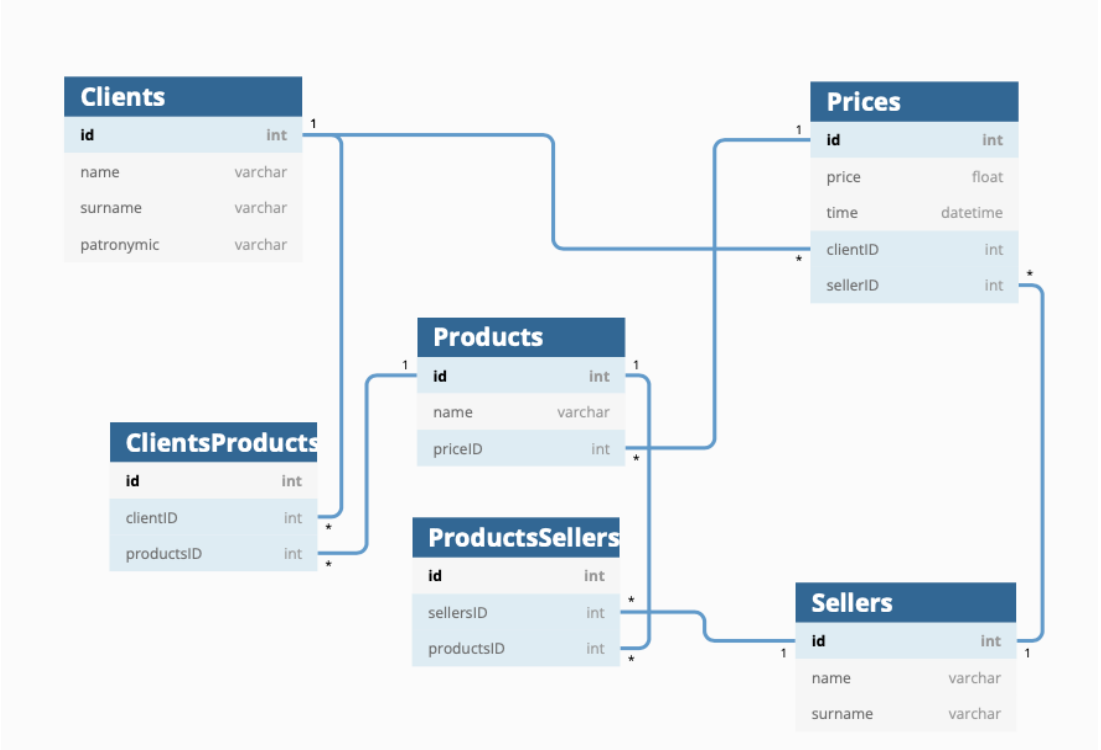


Схема бази даних у графічному вигляді



Опис бази даних

У логічній моделі маємо:

1. Сутність “Продавці” перетворена в таблицю “Sellers ”;
2. Сутність “Клієнти” перетворена в таблицю “Clients”;
3. Сутність “Ціни” перетворена в таблицю “Prices”;
4. Сутність “Продукти” перетворена в таблицю “Products ”;

Для представлення відношення між сутностями “Клієнти” та “Продукти” а також “Продукти” та “Продавці” були створені додаткові таблиці з відповідними назвами “ClientsProducts” та “ProductsSellers” – створені зв'язки типу M:N. Також між сутностями “Клієнти” та “Ціни”, “Ціни” та “Продукти” а також “Продавці” та “Ціни” були використані зв'язки типу 1:N.

Опис меню програми

Меню складається з 9 пунктів:

```
1 => One table
2 => All tables
3 => Insertion
4 => Delete some inf
5 => Updating
6 => Selection
7 => Searching
8 => Random inf
...
0 = > Exit
```

1. One table – вивід на екран однієї таблиці, яку обере користувач.
2. All tables – вивід на екран усіх таблиць.
3. Insertion – вставка у вибрану користувачем таблицю нового рядка.
4. Delete some inf – видалення одного або декількох рядків з обраної таблиці.
5. Updating – оновлення даних у будь-якому рядку, який обере користувач у конкретній таблиці.
6. Selection – формування запитів для фільтрації трьома способами.
7. Random inf – заповнення таблиць випадковими даними.
8. Exit – завершення роботи програми.

Завдання 1

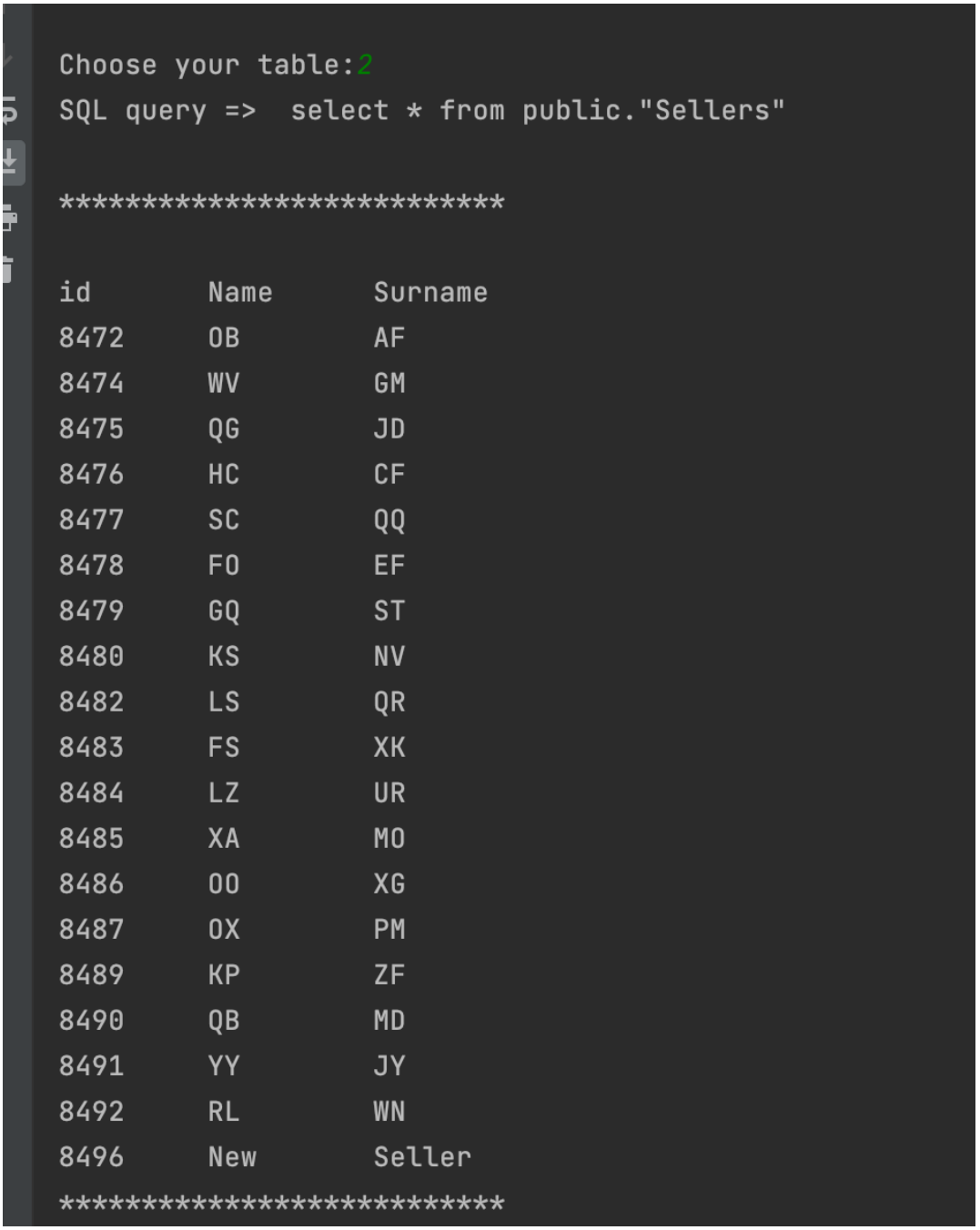
Insert

На прикладі батьківської таблиці Sellers

Запис у Sellers:

```
            1 => One table
            2 => All tables
            3 => Insertion
            4 => Delete some inf
            5 => Updating
            6 => Selection
            7 => Random inf
            ...
            0 = > Exit

Your choice is: 3

        1               => Products
        2               => Sellers
        3               => Clients
        4               => Prices
        5               => ClientsProducts
        6               => SellersProducts

Choose your table: 2
Name = New
Surname = Seller
['NOTICE:  added\n']
1 => Continue insertion, 2 => Stop insertion =>
```

Таблиця Sellers після Insert:



```
Choose your table:2
SQL query =>  select * from public."Sellers"

**************************

id       Name      Surname
8472     OB        AF
8474     WV        GM
8475     QG        JD
8476     HC        CF
8477     SC        QQ
8478     FO        EF
8479     GQ        ST
8480     KS        NV
8482     LS        QR
8483     FS        XK
8484     LZ        UR
8485     XA        MO
8486     OO        XG
8487     OX        PM
8489     KP        ZF
8490     QB        MD
8491     YY        JY
8492     RL        WN
8496     New       Seller
**************************
```

Лістинг для Insert

```
@staticmethod
    def insertProduct(f,s,added,notice):
        connect = Database.connect()
        cursor = connect.cursor()
        insert = 'DO $$ BEGIN if (1=1) and exists (select id from
public."Prices" where id = {}) ' \
                 'then INSERT INTO public."Products"(name, PricesID,
Prices) VALUES ({},{}); ' \
                 'raise notice {}; else raise notice {}; ' \
                 'end if; end $$;'.format(s, f, s, added, notice)
        cursor.execute(insert)
```

```
        connect.commit()
        print(connect.notices)
        cursor.close()
        Database.close(connect)

    @staticmethod
    def insertSeller(f, s, added, notice):
        connect = Database.connect()
        cursor = connect.cursor()
        insert = 'DO $$    BEGIN IF (1=1) THEN ' \
                 'INSERT INTO public."Sellers"(name, surname) values
(\'{}\', \'{}\'); ' \
                 'RAISE NOTICE {};' \
                 ' ELSE RAISE NOTICE {};' \
                 'END IF; ' \
                 'END $$;'.format(f, s, added, notice)
        cursor.execute(insert)
        connect.commit()
        print(connect.notices)
        cursor.close()
        Database.close(connect)

    @staticmethod
    def insertClient(f,s,t,added,notice):
        connect = Database.connect()
        cursor = connect.cursor()
        insert = 'DO $$ BEGIN IF (1=1) THEN ' \
                 'INSERT INTO public."Clients"("Name", "Patronymic",
"Surname") values ({}, {}, {}); ' \
                 'RAISE NOTICE {};' \
                 ' ELSE RAISE NOTICE {};' \
                 'END IF; ' \
                 'END $$;'.format(f,s,t, added, notice)
        cursor.execute(insert)
        connect.commit()
        print(connect.notices)
        cursor.close()
        Database.close(connect)

    @staticmethod
    def insertPrice(f,s,t,p,added,notice):
        connect = Database.connect()
        cursor = connect.cursor()
        insert = 'DO $$    BEGIN IF EXISTS (select "Id" from
public."Clients" where "Id" = {}) and exists (select id from
public."Sellers" where id = {}) THEN ' \
                 'INSERT INTO public."Prices"(time, ClientsID,
SellersID, Price) values (\'{}\', {}, {},{}); ' \
                 'RAISE NOTICE {};' \
                 ' ELSE RAISE NOTICE {};' \
                 'END IF; ' \
                 'END $$;'.format(f, s, t, f, s,p, added, notice)
        cursor.execute(insert)
        connect.commit()
        print(connect.notices)
        cursor.close()
        Database.close(connect)
```

Update

У нашому випадку редагування ключів є неможливим

```
Your choice is: 5

        1                => Products
        2                => Sellers
        3                => Clients
        4                => Prices
        5                => ClientsProducts
        6                => SellersProducts

Choose your table:3
Row to update where id = 35
Name = Upd
Patronymic = Clients
Surname = Record
['NOTICE:  updated\n']
1 => Continue update, 2 => Stop update =>
```

```
        6                => SellersProducts

Choose your table:3
SQL query =>  select * from public."Clients"

**************************

id      Name    Patronymic      Surname
8       VK      XH              NR
24      FN      LH              GC
25      IY      UM              IN
26      ZB      LA              VW
33      IU      HS              YF
36      Testing New             User
35      Upd     Clients         Record
**************************
```

При спробі редагувати рядок, якого не існує:

```
Your choice is: 5

        1               => Products
        2               => Sellers
        3               => Clients
        4               => Prices
        5               => ClientsProducts
        6               => SellersProducts

Choose your table:3
Row to update where id = 32423
Name = Wrong
Patronymic = Rec
Surname = Ord
['NOTICE:  id = 32423 is not present in table.\n']
1 => Continue update, 2 => Stop update =>
```

Лістинг для Update

```
@staticmethod
    def UpdateProduct(idk, name, updated, notice):
        connect = Database.connect()
        cursor = connect.cursor()
        update = 'DO $$ BEGIN IF EXISTS (select id from
public."Products" where id = {}) THEN ' \
                 'update public."Products" set name = {} where id =
{}; ' \
                 'RAISE NOTICE {};' \
                 ' ELSE RAISE NOTICE {};' \
                 'END IF; ' \
                 'END $$;'.format(idk, name, idk, updated, notice)
        cursor.execute(update)
        connect.commit()
        print(connect.notices)
        cursor.close()
        Database.close(connect)


    @staticmethod
    def UpdateSellers(idk, set1, set2, updated, notice):
        connect = Database.connect()
        cursor = connect.cursor()
        update = 'DO $$ BEGIN IF EXISTS (select id from
public."Sellers" where id = {})' \
                 ' THEN ' \
                 'update public."Sellers" set Name = {}, Surname =
{} where id = {}; ' \
```

```
                    'RAISE NOTICE {};' \
                    ' ELSE RAISE NOTICE {};' \
                    'END IF; ' \
                    'END $$;'.format(idk, set1, set2, idk, updated,
notice)
        cursor.execute(update)
        connect.commit()
        print(connect.notices)
        cursor.close()
        Database.close(connect)

    @staticmethod
    def UpdateClients(idk, name, patronymic, surname, updated,
notice):
        connect = Database.connect()
        cursor = connect.cursor()
        update = 'DO $$ BEGIN IF EXISTS (select "Id" from
public."Clients" where "Id" = {}) ' \
                    ' THEN ' \
                    'update public."Clients" set "Name" = {},
"Patronymic" = {}, "Surname" = {} where "Id" = {}; ' \
                    'RAISE NOTICE {};' \
                    ' ELSE RAISE NOTICE {};' \
                    'END IF; ' \
                    'END $$;'.format(idk, name, patronymic, surname,
idk, updated, notice)
        cursor.execute(update)
        connect.commit()
        print(connect.notices)
        cursor.close()
        Database.close(connect)

    @staticmethod
    def UpdatePrices(idk, date, updated, notice):
        connect = Database.connect()
        cursor = connect.cursor()
        update = 'DO $$ BEGIN IF EXISTS (select id from
public."Prices" where id = {}) ' \
                    ' THEN ' \
                    'update public."Prices" set time = \'{}\' where id
= {}; ' \
                    'RAISE NOTICE {};' \
                    ' ELSE RAISE NOTICE {};' \
                    'END IF; ' \
                    'END $$;'.format(idk, date, idk, updated, notice)
        cursor.execute(update)
        connect.commit()
        print(connect.notices)
        cursor.close()
        Database.close(connect)
```

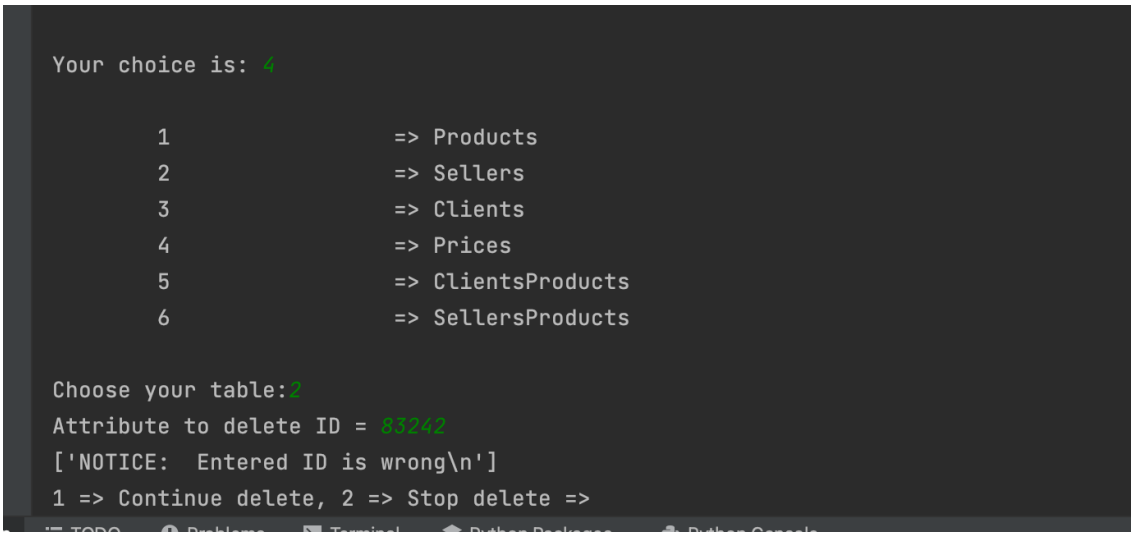
Delete

На прикладі таблиці Sellers (початковий стан таблиці як після Insert)

```
0 = > Exit

Your choice is: 4

        1                   => Products
        2                   => Sellers
        3                   => Clients
        4                   => Prices
        5                   => ClientsProducts
        6                   => SellersProducts

Choose your table:2
Attribute to delete ID = 8492
['NOTICE:  deleted\n']
```

```
Choose your table:2
SQL query =>  select * from public."Sellers"

**************************

id      Name      Surname
8472    OB        AF
8474    WV        GM
8475    QG        JD
8476    HC        CF
8477    SC        QQ
8478    FO        EF
8479    GQ        ST
8480    KS        NV
8482    LS        QR
8483    FS        XK
8484    LZ        UR
8485    XA        MO
8486    OO        XG
8487    OX        PM
8489    KP        ZF
8490    QB        MD
8491    YY        JY
8496    New       Seller
**************************
```

При спробі видалення неіснуючого рядка:



Лістинг для Delete

```
@staticmethod
    def deleteSellers(idk, delete, notice):
        connect = Database.connect()
        cursor = connect.cursor()
        delete = 'DO $$ BEGIN if ' \
                 'exists (select id from public."Sellers" where id
= {}) then ' \
                 'delete from public."SellersProducts" where
SellersID = {};' \
                 'delete from public."Prices" where SellersID =
{};' \
                 'delete from public."Sellers" where id = {};' \
                 'raise notice {};' \
                 'else raise notice {};' \
                 'end if;' \
                 'end $$;'.format(idk, idk, idk, idk, delete,
notice)
        cursor.execute(delete)
        connect.commit()
        print(connect.notices)
        cursor.close()
        Database.close(connect)

    @staticmethod
    def deleteClients(idk, delete, notice):
        connect = Database.connect()
        cursor = connect.cursor()
        delete = 'DO $$ BEGIN if ' \
                 'exists (select "Id" from public."Clients" where
"Id" = {}) then ' \
                 'delete from public."ClientsProducts" where
ClientsID = {};' \
                 'delete from public."Prices" where ClientsID =
{};' \
                 'delete from public."Clients" where "Id" = {};' \
                 'raise notice {};' \
                 'else raise notice {};' \
```

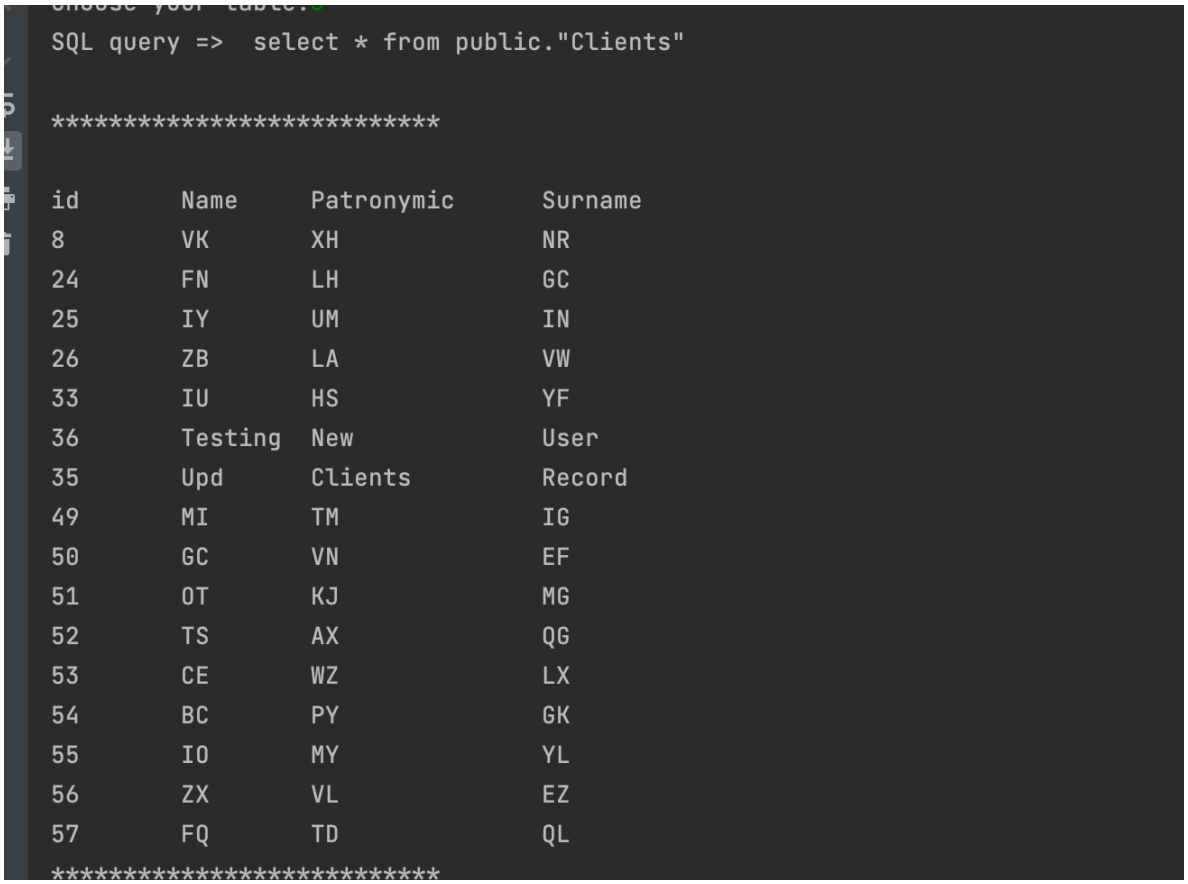
```
                'end if;' \
                'end $$;'.format(idk, idk, idk, idk, delete,
notice)
        cursor.execute(delete)
        connect.commit()
        print(connect.notices)
        cursor.close()
        Database.close(connect)

    @staticmethod
    def deletePrices(idk, delete, notice):
        connect = Database.connect()
        cursor = connect.cursor()
        delete = 'DO $$ BEGIN if exists (select id from
public."Prices" where id = {}) then ' \
                'delete from public."Prices" where id = {};' \
                'raise notice {};' \
                'else raise notice {};' \
                'end if;' \
                'end $$;'.format(idk, idk, delete, notice)
        cursor.execute(delete)
        connect.commit()
        print(connect.notices)
        cursor.close()
        Database.close(connect)
```

Завдання №2

Передбачити автоматичне пакетне генерування “рандомізованих” даних: На прикладі таблиці Clients:

```
Your choice is: 1

        1               => Products
        2               => Sellers
        3               => Clients
        4               => Prices
        5               => ClientsProducts
        6               => SellersProducts

Choose your table:3
SQL query =>  select * from public."Clients"

**************************

id      Name    Patronymic      Surname
8       VK      XH              NR
24      FN      LH              GC
25      IY      UM              IN
26      ZB      LA              VW
33      IU      HS              YF
36      Testing New             User
35      Upd     Clients         Record
**************************
```

```
Your choice is: 7

       1               => Products
       2               => Sellers
       3               => Clients
       4               => Prices
       5               => ClientsProducts
       6               => SellersProducts

Choose your table: 3
How much datas do you want to add => 3
Clients
SQL query =>  INSERT INTO public."Clients"("Name", "Surname", "Patronymic") select chr(trunc(65 + random()*26)::int)||chr(trunc(
Inserted randomly
```



Лістинг

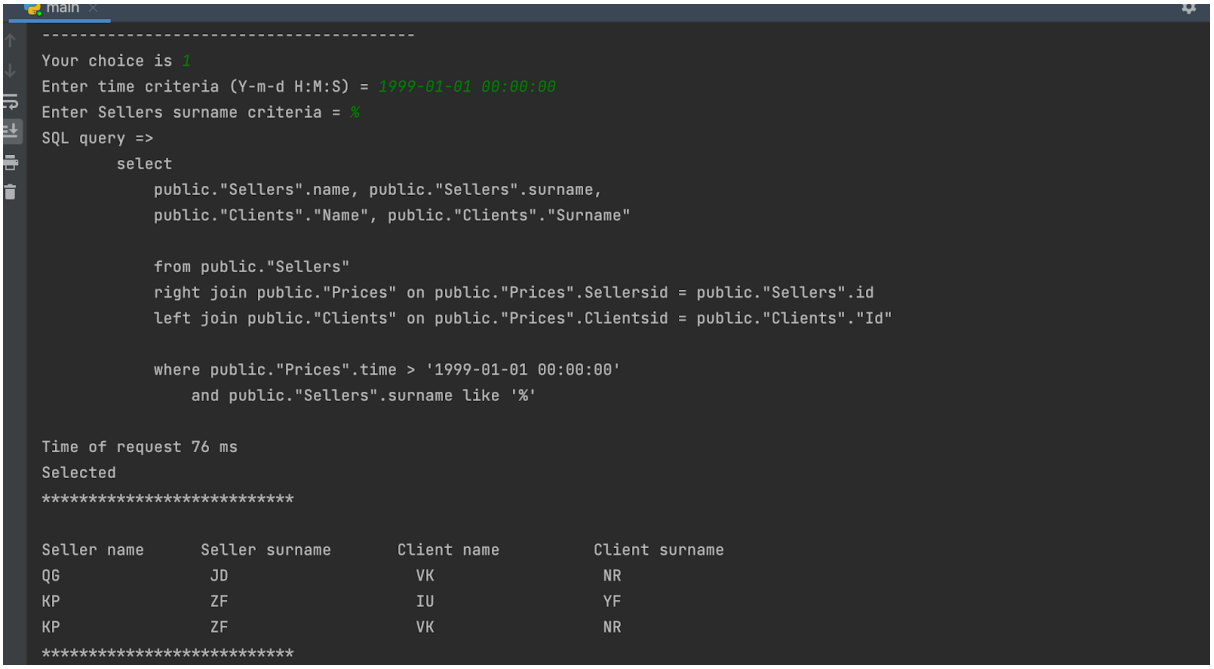
```
        @staticmethod
    def randomik(table, kolvo):
            connect = Database.connect()
            cursor = connect.cursor()
            check = True
            while check:
                if table == 1:
                    res = 0
                    while (True):
                        insert = "INSERT INTO
public.\"Products\"(Name, PricesID) select chr(trunc(65 +
random()*26)::int)||chr(trunc(65 + r" \
                                 "andom()*26)::int)," \
                                 "(select id from public.\"Prices\"
order by random() limit 1)" \
                                 " from
generate_series(1,{})".format(kolvo)
                        cursor.execute(insert)
                        res = res + 1
                        if(res == int(kolvo)):
                            break
                    check = False
                elif table == 2:
                    res = 0
                    while (True):
                        insert = "INSERT INTO
public.\"Sellers\"(Name, Surname) select chr(trunc(65 +
```

```
random()*26)::int)||chr(trunc(65 + r" \
                            "andom()*26)::int), " \
                            "chr(trunc(65 +
random()*26)::int)||chr(trunc(65 + random()*26)::int) " \
                            "from
generate_series(1,{})".format(kolvo)
                        cursor.execute(insert)
                        res = res + 1
                        if(res == int(kolvo)):
                            break
                    check = False
                elif table == 3:
                    res = 0
                    while (True):
                        insert = "INSERT INTO
public.\"Clients\"(\"Name\", \"Surname\", \"Patronymic\") select
chr(trunc(65 + random()*26)::int)||chr(trunc(65 + r" \
                            "andom()*26)::int), " \
                            "chr(trunc(65 +
random()*26)::int)||chr(trunc(65 + random()*26)::int), " \
                            "chr(trunc(65 +
random()*26)::int)||chr(trunc(65 + random()*26)::int)" \
                            "from
generate_series(1,{})".format(kolvo)
                        cursor.execute(insert)
                        res = res + 1
                        if(res == int(kolvo)):
                            break
                    check = False
                elif table == 4:
                    res = 0
                    while (True):
                        insert = "INSERT INTO
public.\"Prices\"(Time, ClientsID, SellersID) values(" \
                                "(select NOW() + (random() *
(NOW()+'90 days' - NOW())) + '{} days')," \
                                "(select \"Id\" from
public.\"Clients\" order by random() limit 1)," \
                                "(select id from
public.\"Sellers\" order by random() limit 1))".format(kolvo)
                        cursor.execute(insert)
                        res = res + 1
                        if (res == int(kolvo)):
                            break
                    check = False
                elif table == 5:
                    res = 0
                    while (True):
                        insert = "INSERT INTO
public.\"ClientsProducts\"(ClientsID, ProductsID) values(" \
                                "(select \"Id\" from
public.\"Clients\" order by random() limit 1)," \
                                "(select id from
public.\"Products\" order by random() limit 1))".format(kolvo)
                        cursor.execute(insert)
                        res = res + 1
                        if (res == int(kolvo)):
```

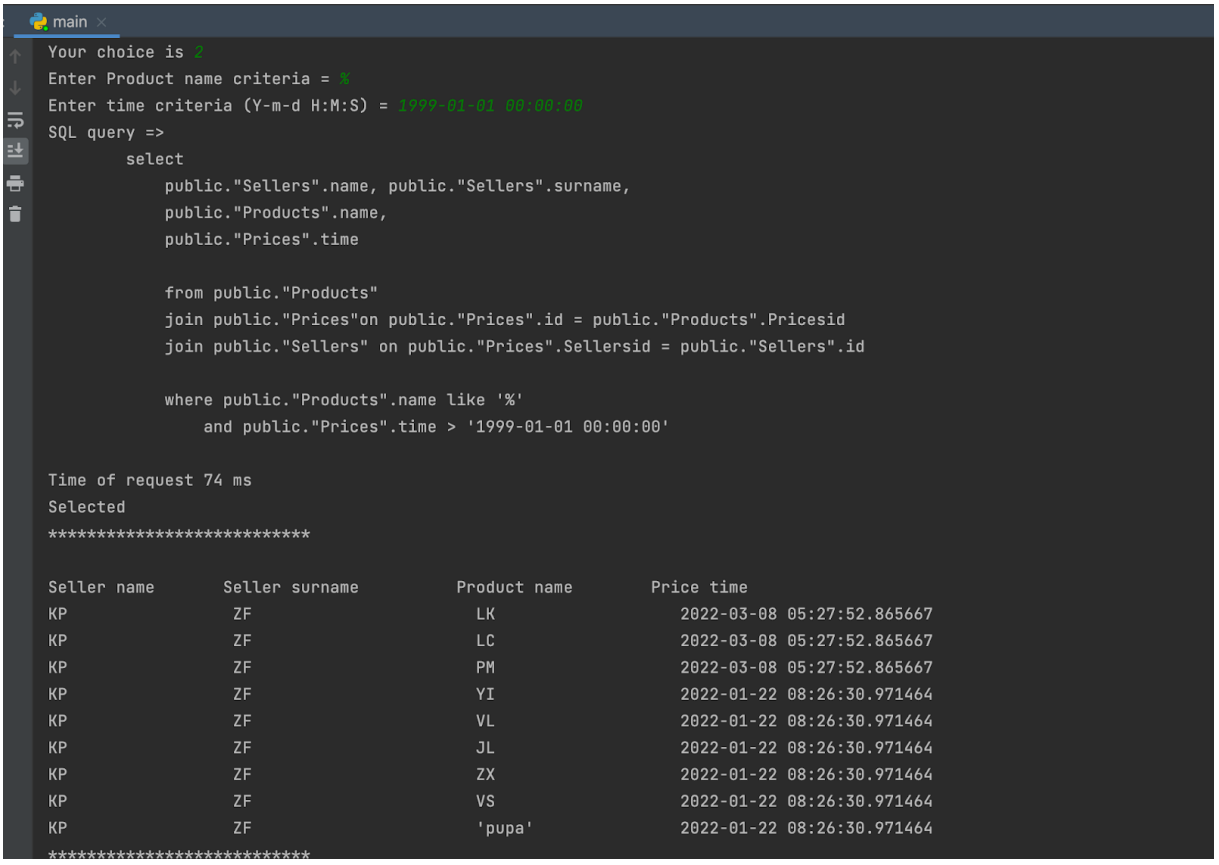
```
                            break
                    check = False
                elif table == 6:
                    res = 0
                    while (True):
                        insert = "INSERT INTO
public.\"SellersProducts\"(SellersID, ProductsID) values(" \
                                 "(select id from
public.\"Sellers\" order by random() limit 1)," \
                                 "(select id from
public.\"Products\" order by random() limit 1))".format(kolvo)
                        cursor.execute(insert)
                        res = res + 1
                        if (res == int(kolvo)):
                            break
                    check = False
            print(Tables[table])
            print("SQL query => ", insert)
            connect.commit()
            print('Inserted randomly')
            cursor.close()
            Database.close(connect)
```

Завдання №3

Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно.



```
main
--------------------------------------
Your choice is 1
Enter time criteria (Y-m-d H:M:S) = 1999-01-01 00:00:00
Enter Sellers surname criteria = %
SQL query =>
        select
            public."Sellers".name, public."Sellers".surname,
            public."Clients"."Name", public."Clients"."Surname"

            from public."Sellers"
            right join public."Prices" on public."Prices".Sellersid = public."Sellers".id
            left join public."Clients" on public."Prices".Clientsid = public."Clients"."Id"

            where public."Prices".time > '1999-01-01 00:00:00'
                and public."Sellers".surname like '%'

Time of request 76 ms
Selected
**************************

Seller name      Seller surname       Client name          Client surname
QG                JD                   VK                   NR
KP                ZF                   IU                   YF
KP                ZF                   VK                   NR
**************************
```



```
main
Your choice is 2
Enter Product name criteria = %
Enter time criteria (Y-m-d H:M:S) = 1999-01-01 00:00:00
SQL query =>
        select
            public."Sellers".name, public."Sellers".surname,
            public."Products".name,
            public."Prices".time

            from public."Products"
            join public."Prices"on public."Prices".id = public."Products".Pricesid
            join public."Sellers" on public."Prices".Sellersid = public."Sellers".id

            where public."Products".name like '%'
                and public."Prices".time > '1999-01-01 00:00:00'

Time of request 74 ms
Selected
**************************

Seller name       Seller surname          Product name        Price time
KP                 ZF                      LK                   2022-03-08 05:27:52.865667
KP                 ZF                      LC                   2022-03-08 05:27:52.865667
KP                 ZF                      PM                   2022-03-08 05:27:52.865667
KP                 ZF                      YI                   2022-01-22 08:26:30.971464
KP                 ZF                      VL                   2022-01-22 08:26:30.971464
KP                 ZF                      JL                   2022-01-22 08:26:30.971464
KP                 ZF                      ZX                   2022-01-22 08:26:30.971464
KP                 ZF                      VS                   2022-01-22 08:26:30.971464
KP                 ZF                      'pupa'               2022-01-22 08:26:30.971464
**************************
```

```
--------------------------------------
Your choice is 3
Enter Sellers surname criteria = %
Enter Clients surname criteria = %
SQL query =>
       select public."Products".name from public."Products"

           join public."Prices"on public."Prices".id = public."Products".Pricesid

           join public."SellersProducts"
               on public."SellersProducts".Productsid = public."Products".id
               and public."SellersProducts".Sellersid = public."Prices".Sellersid

           join public."ClientsProducts"
               on public."ClientsProducts".Productsid = public."Products".id
               and public."ClientsProducts".Clientsid = public."Prices".Clientsid

           join public."Clients" on "ClientsProducts".Clientsid = public."Clients"."Id"
           join public."Sellers" on public."SellersProducts".Sellersid = public."Sellers".id

           where public."Sellers".surname like '%'
               and public."Clients"."Surname" like '%'

           group by public."Products".name

Time of request 75 ms
Selected
*************************

Name
PM
*************************
```

Лістинг

```
   @staticmethod
    def selectionone(timestamp, SellerSurname):
        connect = Database.connect()
        cursor = connect.cursor()
        select = """
        select
            public."Sellers".name, public."Sellers".surname,
            public."Clients"."Name", public."Clients"."Surname"

            from public."Sellers"
            right join public."Prices" on public."Prices".Sellersid
= public."Sellers".id
            left join public."Clients" on public."Prices".Clientsid
= public."Clients"."Id"

            where public."Prices".time > '{}'
                and public."Sellers".surname like '{}'
        """.format(timestamp, SellerSurname)
        print("SQL query => ", select)
        beg = int(time.time() * 1000)
        cursor.execute(select)
        end = int(time.time() * 1000) - beg
        datas = cursor.fetchall()
        print('Time of request {} ms'.format(end))
        print('Selected')
        cursor.close()
        Database.close(connect)
        return datas
```

```
    @staticmethod
    def selectiontwo(subj, PriceTime):
        connect = Database.connect()
        cursor = connect.cursor()
        select = """
        select
            public."Sellers".name, public."Sellers".surname,
            public."Products".name,
            public."Prices".time

            from public."Products"
            join public."Prices"on public."Prices".id =
public."Products".Pricesid
            join public."Sellers" on public."Prices".Sellersid =
public."Sellers".id

            where public."Products".name like '{}'
                and public."Prices".time > '{}'
        """.format(subj, PriceTime)
        print("SQL query => ", select)
        beg = int(time.time() * 1000)
        cursor.execute(select)
        end = int(time.time() * 1000) - beg
        datas = cursor.fetchall()
        print('Time of request {} ms'.format(end))
        print('Selected')
        cursor.close()
        Database.close(connect)
        return datas

    @staticmethod
    def selectionthree(SellersS, ClientsS):
        connect = Database.connect()
        cursor = connect.cursor()
        select = """
        select public."Products".name from public."Products"

            join public."Prices"on public."Prices".id =
public."Products".Pricesid

            join public."SellersProducts"
                on public."SellersProducts".Productsid =
public."Products".id
                and public."SellersProducts".Sellersid =
public."Prices".Sellersid

            join public."ClientsProducts"
                on public."ClientsProducts".Productsid =
public."Products".id
                and public."ClientsProducts".Clientsid =
public."Prices".Clientsid

            join public."Clients" on "ClientsProducts".Clientsid =
public."Clients"."Id"
            join public."Sellers" on
public."SellersProducts".Sellersid = public."Sellers".id

            where public."Sellers".surname like '{}'
```

```
                and public."Clients"."Surname" like '{}'

            group by public."Products".name
        """.format(SellersS, ClientsS)
        print("SQL query => ", select)
        beg = int(time.time() * 1000)
        cursor.execute(select)
        end = int(time.time() * 1000) - beg
        datas = cursor.fetchall()
        print('Time of request {} ms'.format(end))
        print('Selected')
        cursor.close()
        Database.close(connect)
        return datas

@staticmethod
    def selectionone(timestamp, teacherSurname):
        connect = Database.connect()
        cursor = connect.cursor()
        select = """
        select
            public."Teachers".name, public."Teachers".surname,
            public."Pupils"."Name", public."Pupils"."Surname"

            from public."Teachers"
            right join public."Marks" on public."Marks".teachersid
= public."Teachers".id
            left join public."Pupils" on public."Marks".pupilsid =
public."Pupils"."Id"

            where public."Marks".time > '{}'
                and public."Teachers".surname like '{}'
        """.format(timestamp, teacherSurname)
        print("SQL query => ", select)
        beg = int(time.time() * 1000)
        cursor.execute(select)
        end = int(time.time() * 1000) - beg
        datas = cursor.fetchall()
        print('Time of request {} ms'.format(end))
        print('Selected')
        cursor.close()
        Database.close(connect)
        return datas

    @staticmethod
    def selectiontwo(subj, markTime):
        connect = Database.connect()
        cursor = connect.cursor()
        select = """
        select
            public."Teachers".name, public."Teachers".surname,
            public."Subjects".name,
            public."Marks".time

            from public."Subjects"
            join public."Marks"on public."Marks".id =
public."Subjects".marksid
            join public."Teachers" on public."Marks".teachersid =
```

```
public."Teachers".id

            where public."Subjects".name like '{}'
                and public."Marks".time > '{}'
        """.format(subj, markTime)
        print("SQL query => ", select)
        beg = int(time.time() * 1000)
        cursor.execute(select)
        end = int(time.time() * 1000) - beg
        datas = cursor.fetchall()
        print('Time of request {} ms'.format(end))
        print('Selected')
        cursor.close()
        Database.close(connect)
        return datas

    @staticmethod
    def selectionthree(teachersS, pupilsS):
        connect = Database.connect()
        cursor = connect.cursor()
        select = """
        select public."Subjects".name from public."Subjects"

            join public."Marks"on public."Marks".id =
public."Subjects".marksid

            join public."TeachersSubjects"
                on public."TeachersSubjects".subjectsid =
public."Subjects".id
                and public."TeachersSubjects".teachersid =
public."Marks".teachersid

            join public."PupilsSubjects"
                on public."PupilsSubjects".subjectsid =
public."Subjects".id
                and public."PupilsSubjects".pupilsid =
public."Marks".pupilsid

            join public."Pupils" on "PupilsSubjects".pupilsid =
public."Pupils"."Id"
            join public."Teachers" on
public."TeachersSubjects".teachersid = public."Teachers".id

            where public."Teachers".surname like '{}'
                and public."Pupils"."Surname" like '{}'

            group by public."Subjects".name
        """.format(teachersS, pupilsS)
        print("SQL query => ", select)
        beg = int(time.time() * 1000)
        cursor.execute(select)
        end = int(time.time() * 1000) - beg
        datas = cursor.fetchall()
        print('Time of request {} ms'.format(end))
        print('Selected')
        cursor.close()
        Database.close(connect)
        return datas
```

Методи моделі

existingtable() – перевірка на існування таблиці.
Outputonetable() – вивід вибраної таблиці.
Insert<tablename>() – додавання рядків у таблицях.
Update<tablename>() – оновлення рядків у таблилях.
Delete<tablename>() – видалення рядків у таблицях.
selectionone() – пошуковий запис 1.
selectiontwo() – пошуковий запис 2. selectionthree() – пошуковий запис 3.
randomik() – заповнення таблиць випадковими даними.