

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**

*на тему: “ Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL ”*

Виконав:

студент ІII курсу

групи КВ-91

Батура В. В.

Перевірив:

Київ – 2021

# Постановка задачі

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

*Загальне завдання* роботи полягає у наступному:

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

1. Програмний код виконати згідно шаблону MVC (модель - подання - контролер).

**Інформація про програму**

Посилання на репозиторій у GitHub з звітом 1 лабораторної роботи: https://github.com/BaturaVetal/database

Використана мова програмування: Python 3.9

Використані бібліотеки: psycopg2 (для зв’язку з СУБД), datetime (для роботи з датою і передачею її у запити до БД), time (для виміру часу запиту пошуку для завдання 3), sys (для реалізації консольного інтерфейсу).

# Відомості про обрану предметну галузь з лабораторної роботи №1

Обрана предметна галузь передбачає отримання і обробку замовлень з різних інтернет-магазинів. У обраній базі даних зберігається інформація про замовлення (таблиця Order), що надійшли до магазинів (таблиця Shop), каталог товарів (таблиця Product) яких розміщено у сутності Catalog. Виходячи з даних, що зберігаються в базі, можна визначити: коли було зроблено замовлення, ким воно було зроблено, які товари до нього входять, до якого саме з магазинів мережі надійшло. Відповідно, кожен товар відноситься до певного каталогу з дерева каталогів, а також має інші характеристики. Товар зв’язаний зовнішнім ключем з замовленням та каталогом, каталог та замовлення в свою чергу зв’язані з магазином.

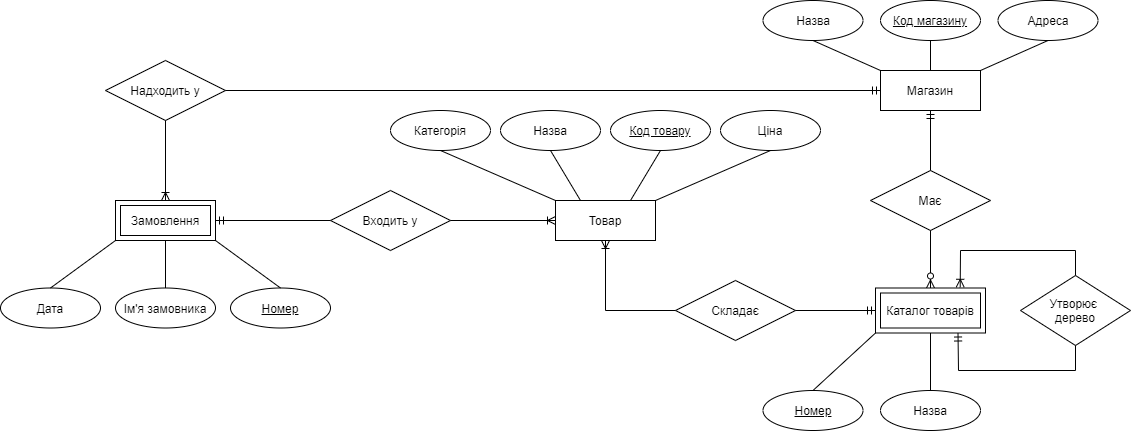


Рисунок 1. ER-діаграма, побудована за нотацією Чена

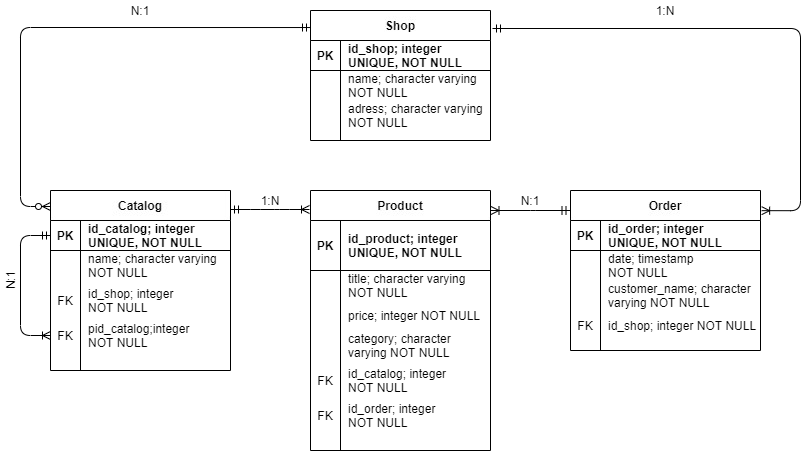


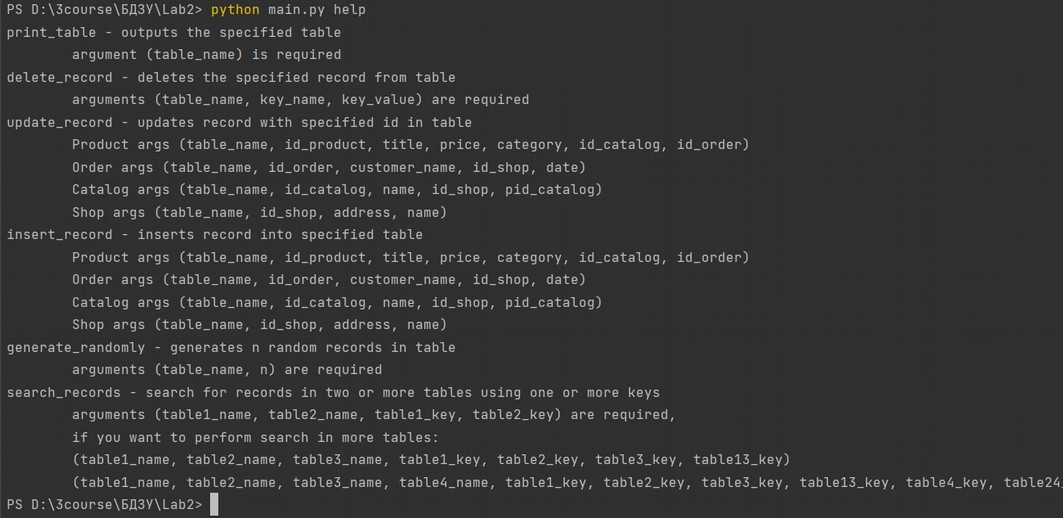
Рисунок 2. Схема бази даних

Таблиця 1. Опис структури БД

|  |  |  |
| --- | --- | --- |
| Сутність | Атрибут | Тип атрибуту |
| **Product** - *містить дані про замовлений товар* | **id\_product** *- унікальний ідентифікатор товару* **title** *- назва товару* **price** *- ціна товару*  **category** *- категорія, до якої належить товар*  **id\_catalog** *- ідентифікатор каталогу, в якому знаходиться товар*  **id\_order** *- ідентифікатор замовлення, до якого додано товар* | **integer** *(числовий)*  **character varying** *(рядок)* **integer** *(числовий)* **character varying** *(рядок)*  **integer** *(числовий)*  **integer** *(числовий)* |
| **Catalog** - *містить у собі всі товари магазину* | **id\_catalog** *- унікальний ідентифікатор каталогу*  **name** *- назва каталогу*  **id\_shop** *- ідентифікатор магазину, до якого відноситься даний каталог* **pid\_catalog** *- посилання на батьківський каталог* | **integer** *(числовий)*  **character varying** *(рядок)*  **integer** *(числовий)*  **integer** *(числовий)* |
| **Order** - *зберігає дані про замовлення* | **id\_order** *- унікальний ідентифікатор замовлення*  **customer\_name** *- ім’я замовника*  **date** *- мітка часу, коли було оформлено замовлення*  **id\_shop** *- ідентифікатор магазину, в який надійшло замовлення* | **integer** *(числовий)*  **character varying** *(рядок)* **timestamp without time zone** *(мітка часу)*  **integer** *(числовий)* |
| **Shop** - *містить інформацію про магазин* | **id\_shop** *- унікальний ідентифікатор магазину*  **name** *- назва магазину*  **address** *- фізична адреса магазину* | **integer** *(числовий)*  **character varying** *(рядок)*  **character varying** *(рядок)* |

# Схема меню користувача

На знімку екрану терміналу продемонстовано виконання команди help, що показує усі доступні користувачу команди, коротко описує їх та надає список обов’язкових аргументів. Кожна команда запускає відповідний метод об’єкту класу Controller, який реалізує передачу аргументів у клас View на перевірку і за умови їх коректності, Controller далі передає ці аргументи у клас Model, що здійснює запит до бази даних.



# Завдання 1

***Методи реалізовані до 1 завдання:***

print\_table – за умови коректності імені таблиці виводить вміст цієї таблиці у вікно терміналу. Аргументом може бути одне із імен: Product, Order, Catalog, Shop.

delete\_record – за умови правильності введених аргументів, наявності відповідного запису (з вказаним значенням первинного ключа) та незалежності інших таблиць від цього запису (до цього запису немає зовнішнього ключа з іншої таблиці), видаляє запис з вказаним первинним ключем. Аргументами є table\_name, key\_name, key\_value

update\_record – за умови правильності введених аргументів, наявності відповідного запису (з вказаним значенням первинного ключа) та записів інших таблиць (на які хочемо змінити поточні), змінює усі поля, окрім первинного ключа у обраному записі.

Аргументи для кожної таблиці:

Product id\_product(int) title(str) price(float) category(str) id\_catalog(int) id\_order(int)

Order id\_order(int) customer\_name(str) id\_shop(int) date(datetime)

Catalog id\_catalog(int) name(str) id\_shop(int) pid\_catalog(int)

Shop id\_shop(int) address(str) name(str)

insert\_record – за умови правильності введених аргументів, відсутності відповідного запису (з вказаним значенням первинного ключа) та наявності записів інших таблиць (на які хочемо посилатись зі створеного запису), вставляє новий рядок у таблицю з обраними значеннями полів.

Аргументи для кожної таблиці:

Product id\_product(int) title(str) price(float) category(str) id\_catalog(int) id\_order(int)

Order id\_order(int) customer\_name(str) id\_shop(int) date(datetime)

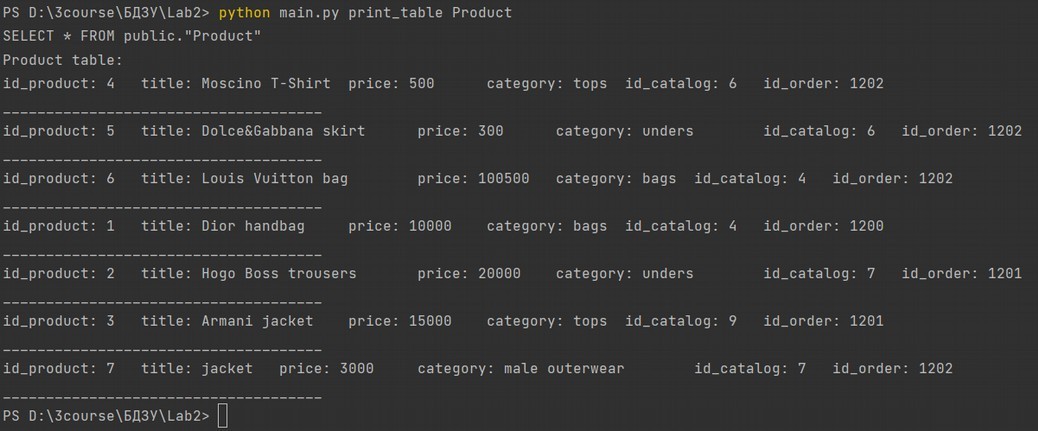
Catalog id\_catalog(int) name(str) id\_shop(int) pid\_catalog(int)

Shop id\_shop(int) address(str) name(str)

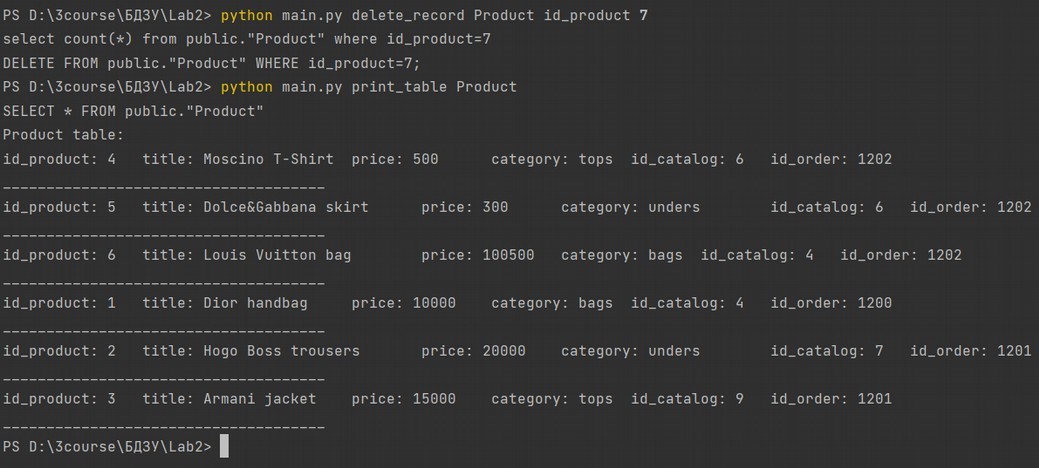
## Запит на видалення

Для перевірки роботи розглянемо запити на видалення дочірньої таблиці Product та батьківської таблиці Order.

Таблиця Product до видалення запису 7:



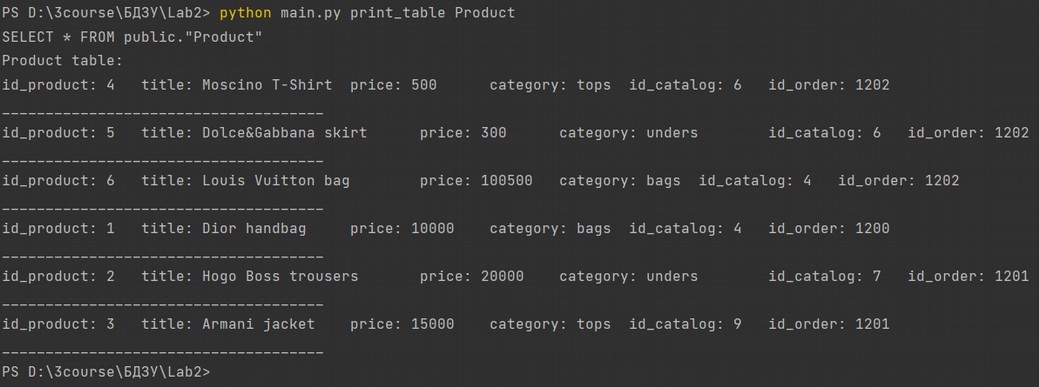
Таблиця Product після видалення запису 7:



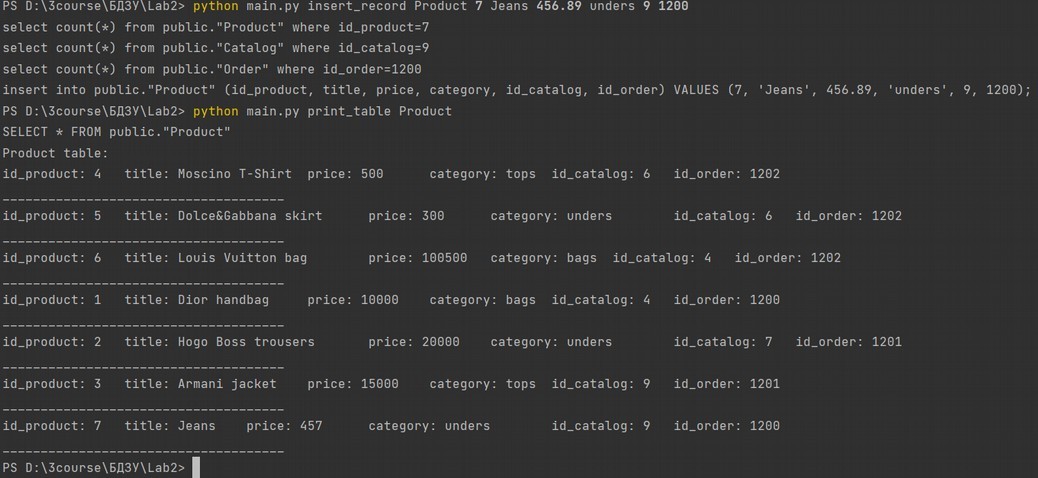
## Запит на вставку поля

Для перевірки роботи розглянемо запити на вставки в дочірню таблицю Product.

Таблиця Product до вставки запису 7:



Таблиця Product після вставки запису 7:



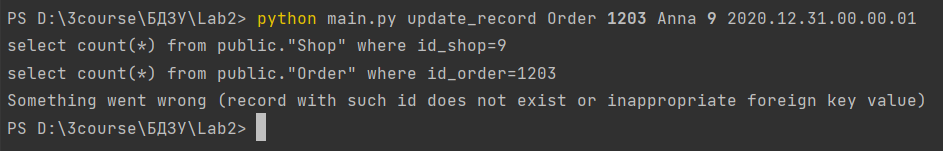
## Запит на зміну полів

Для перевірки роботи розглянемо запити на зміну з неіснуючим значенням зовнішнього ключа батьківської таблиці Shop.

Записи у батьківській таблиці Shop:



Спроба зміни запису у дочірній таблиці Order з неіснуючим зовнішнім ключем 9:



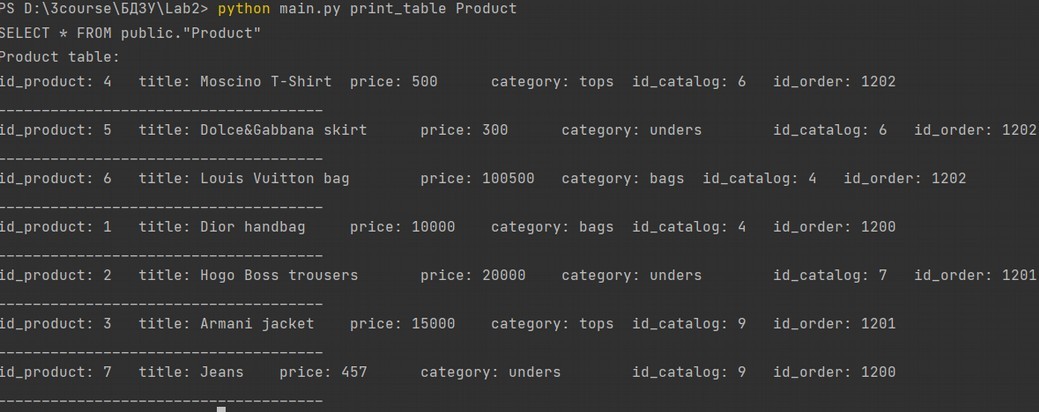
# Завдання 2

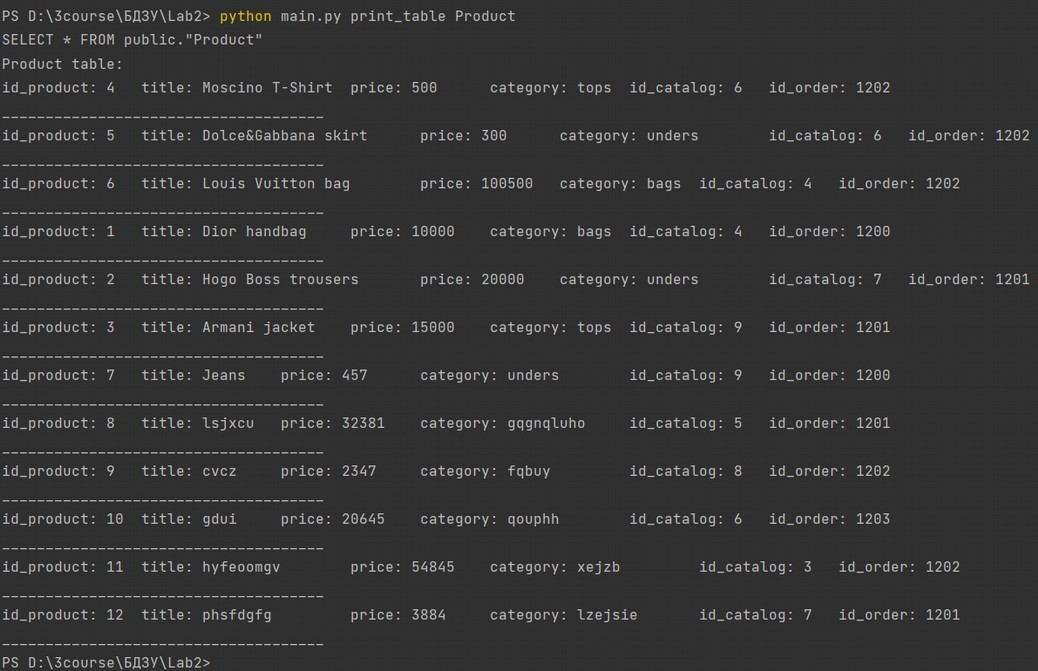
## Метод реалізований до 2 завдання:

generate\_randomly – за умови введення правильного імені таблиці та числа n відмінного від нуля, здійснює генерування n псевдорандомізованих записів у обраній таблиці. Аргементами є ім’я таблиці та часло записів, що мають бути створені.

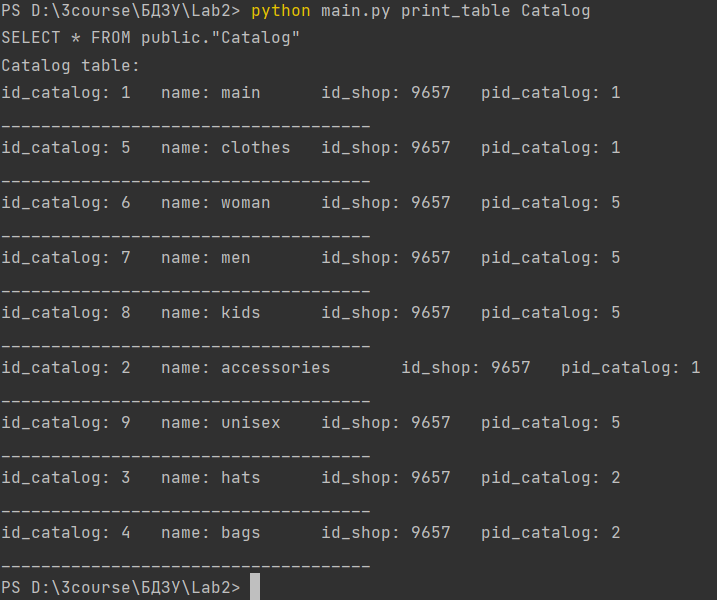
Вставка 5 псевдорандомізованих записів у кожну з таблиць.

Початкова таблиця Product:



Модифікована таблиця Product:

Початкова таблиця Catalog:

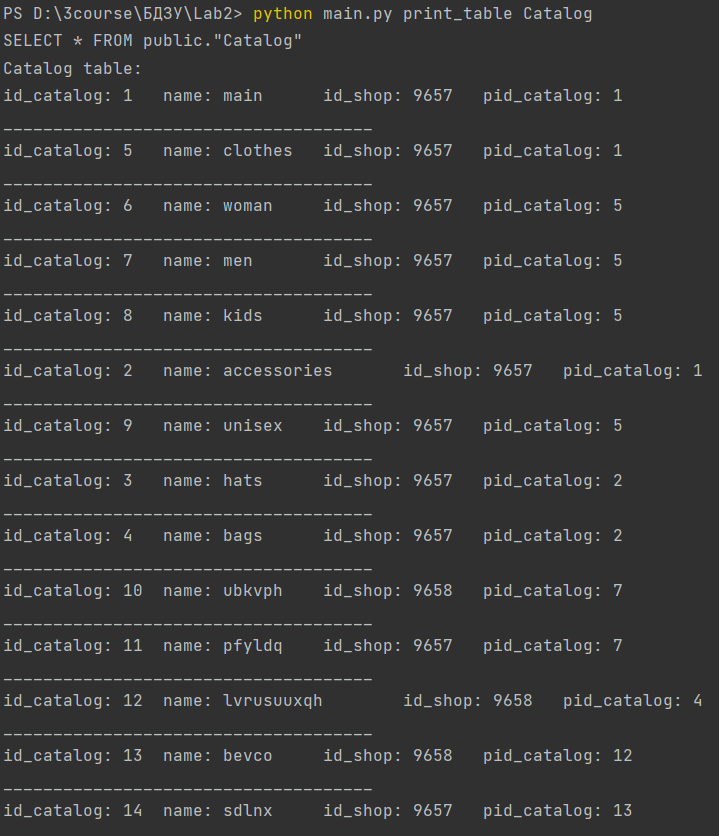


Запит:

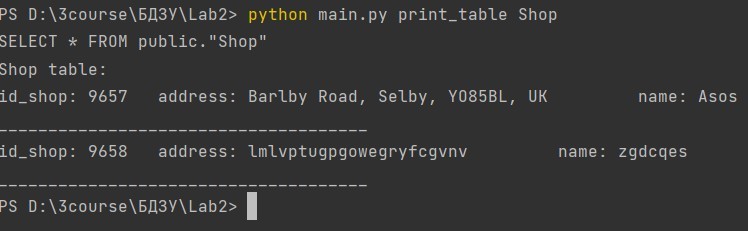
PS D:\3course\БДЗУ\Lab2> python main.py generate\_randomly Catalog 5

insert into public."Catalog" select (SELECT MAX(id\_catalog)+1 FROM public."Catalog"), array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) FROM generate\_series(1, FLOOR(RANDOM()\*(15-5)+5):: integer)), ''), (SELECT id\_shop FROM public."Shop" LIMIT 1 OFFSET (round(random() \*((SELECT COUNT(id\_shop) FROM public."Shop")-1)))), (SELECT id\_catalog FROM public."Catalog" LIMIT 1 OFFSET (round(random() \* ((SELECT COUNT(id\_catalog) FROM public."Catalog")-1))));

Модифікована таблиця Catalog:



Початкова таблиця Shop:

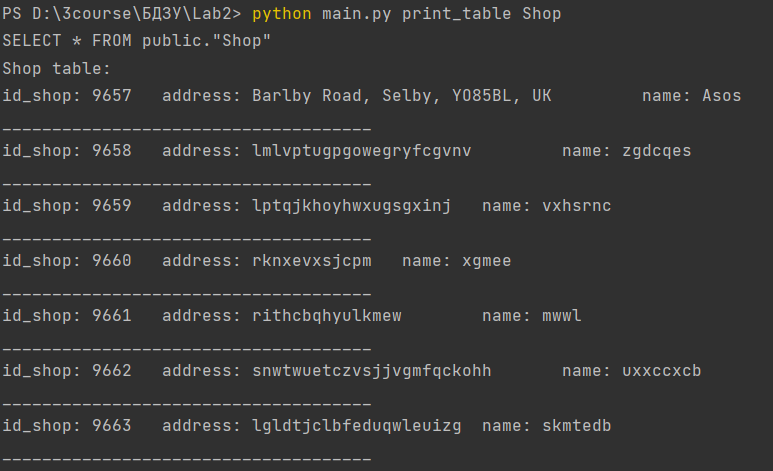


Запит:

PS D:\3course\БДЗУ\Lab2> python main.py generate\_randomly Shop 5

insert into public."Shop" select (SELECT MAX(id\_shop)+1 FROM public."Shop"), array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) FROM generate\_series(1, FLOOR(RANDOM()\*(25-10)+10):: integer)), ''), array\_to\_string(ARRAY(SELECT chr((97 + round(random()\* 25)) :: integer) FROM generate\_series(1, FLOOR(RANDOM()\*(10-4)+4):: integer)), '');

Модифікована таблиця Shop:



# Завдання 3

## Метод реалізований до 3 завдання:

search\_records – за умови введення потрібної кількості аргументів та правильного задання умов пошуку, реалізує пошук за 1 та більше атрибутами з вказаних таблиць (від двох до чотирьох) і виводить у вікно терміналу результат пошуку (або нічого, якщо пошук не дав результатів) та час, за який було проведено запит. Початково потрібно вказати агрументи:

table1\_name table2\_name table1\_key table2\_key aбо

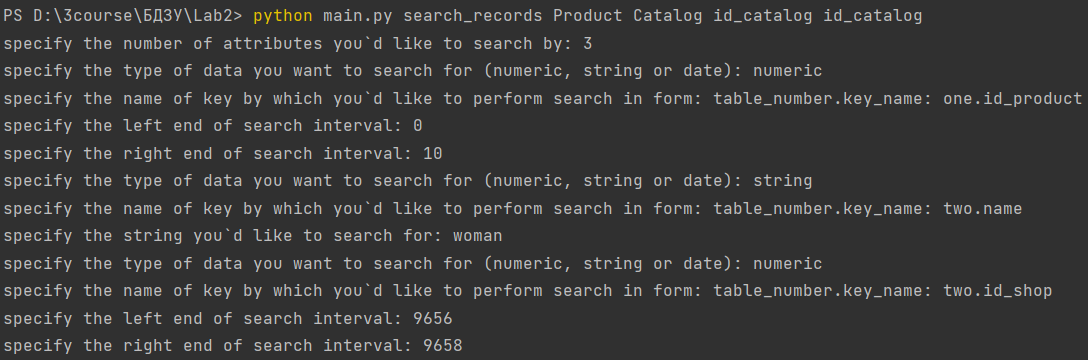
table1\_name table2\_name table3\_name table1\_key table2\_key table3\_key table13\_key або

table1\_name table2\_name table3\_name table4\_name table1\_key table2\_key table3\_key table13\_key table4\_key table24\_key

де table13\_key, table24\_key – це зовнішні ключі, що зв’язують 1 та 3 таблицю, або 2 та 4.

Після вказання цієї інформації потрібно буде вказати кількість атрибутів для пошуку, тип пошуку, ім’я атрибуту (обов’язково з вказанням до якої таблиці з перелічених аргументів він відноситься: one.key\_name, two.key\_name, three.key\_name або four.key\_name), та значення (спочатку лівий кінець інтервалу, потім правий для числового пошуку та пошуку за датою, або рядок для пошуку за ключовим словом). Спочатку вказуються всі дані для першого атрибуту, потім для другого і т.д. до введеної кількості атрибутів.

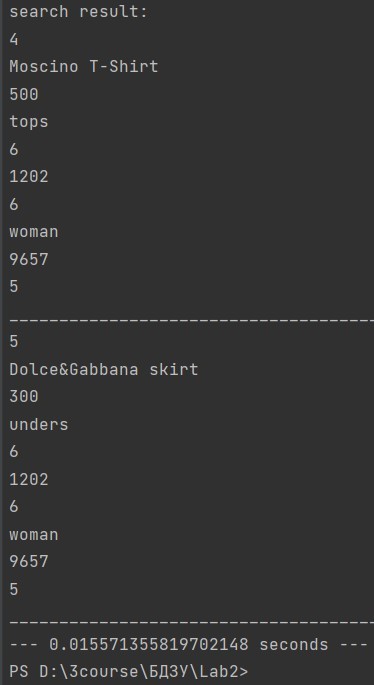
Пошук за трьома атрибутами з двох таблиць (Product, Catalog). Формування запиту:



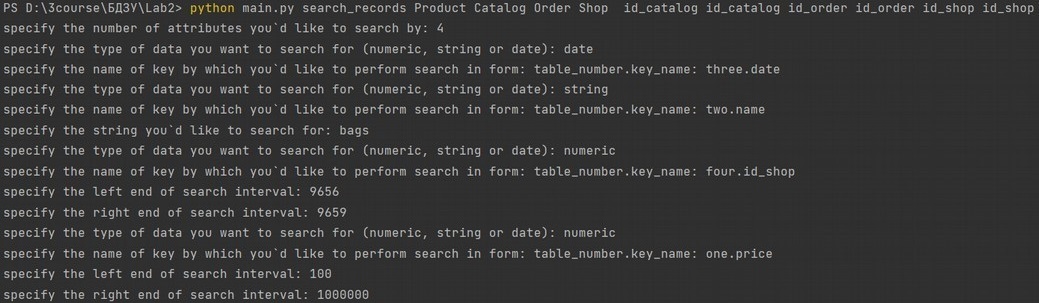
Запит:

select \* from public."Product" as one inner join public."Catalog" as two on one."id\_catalog"=two."id\_catalog" where 0<one.id\_product and one.id\_product<10 and two.name LIKE 'woman' and 9656<two.id\_shop and two.id\_shop<9658

Результат:



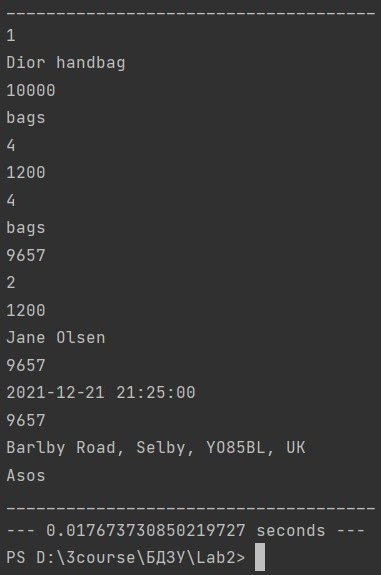
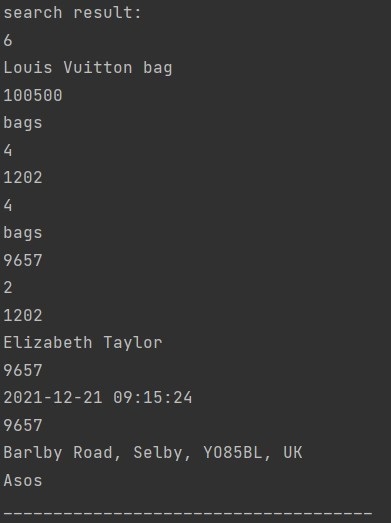
Пошук за чотирма атрибутами з чотирьох таблиць (Product, Catalog, Order, Shop) Формування запиту:



Запит:

select \* from public."Product" as one inner join public."Catalog" as two on one."id\_catalog"=two."id\_catalog" inner join public."Order" as three on three."id\_order"=one."id\_order" inner join public."Shop" as four on four."id\_shop"=two."id\_shop"where three.date BETWEEN '2021-12-01 00:00:00' AND '2021- 12-30 00:00:00' and two.name LIKE 'bags' and 9656<four.id\_shop and four.id\_shop<9659 and 100<one.price and one.price<1000000

Результат:



# Завдання 4

Даний модуль є точкою доступу до бази даних з програми. Саме в ньому реалізуються всі запити. Для цього в ньому використовується бібліотека мови Python – psycopg2.

Конструктор класу Model налагоджує зв’язок із сервером і видає повідомлення про помилку, якщо зв’язок не було встановлено.

Методи request, get, get\_el здійснюють запити до бази даних за допомогою cursor вони всі повертають False якщо виникла помилка і запит не відбувся, а якщо вдалося зробити запит, request повертає True, get повертає усі дані що було взято з запитів SELECT (масив кортежів з записами таблиць), get\_el повертає тільки перший запис.

Метод count повертає кількість усіх записів таблиці.

Метод find повертає кількість записів таблиці, що відповідають певній умові (або False, якщо записів не знайдено).

Методи max, min повертають відповідно максимальне і мінімальне значення зазначеного ключа у таблиці.

Методи print\_products, print\_order, print\_catalog, print\_shop здійснюють отримання з БД та виведення на екран відповідних таблиць.

Метод delete\_data реалізує запит на видалення відповідного запису таблиці.

Методи update\_data\_(table name) відправляють до БД запит на оновлення даних у певному полі таблиці.

Методи insert\_data\_(table name) відправляють до БД запит на вставку запису в таблицю.

Методи (table name)\_data\_generator реалізують запит до БД на вставку псевдорандомізованих даних (для первинного ключа максимальне вже існуюче значення +1, для зовнішніх ключів — одне із можливих значень для цього ключа).

Методи search\_data\_(number)\_tables реалізують запит на отримання результату пошуку серед number таблиць за рядком, що генерується в контроллері методами: numeric\_search, string\_search, date\_search.

Методи numeric\_search, string\_search, date\_search приймають на вхід параметри пошуку (діапазони, рядковий шаблон, ключі) та повертають рядок пошуку за відповідним типом атрибуту.

**Код програмного модулю “model.py”**

import datetime

import psycopg2 as ps

class Model:

def init (self): self.conn = None try:

self.conn = ps.connect( database="postgres", user='postgres', password="\*\*\*\*\*\*\*\*", host='127.0.0.1',

port="5432",

)

except(Exception, ps.DatabaseError) as error:

print("[INFO] Error while working with Postgresql", error)

def request(self, req: str): try:

cursor = self.conn.cursor() print(req) cursor.execute(req) self.conn.commit()

return True

except(Exception, ps.DatabaseError, ps.ProgrammingError) as error: print(error)

self.conn.rollback() return False

def get(self, req: str): try:

cursor = self.conn.cursor() print(req) cursor.execute(req) self.conn.commit()

return cursor.fetchall()

except(Exception, ps.DatabaseError, ps.ProgrammingError) as error: print(error)

self.conn.rollback() return False

def get\_el(self, req: str): try:

cursor = self.conn.cursor() print(req) cursor.execute(req) self.conn.commit()

return cursor.fetchone()

except(Exception, ps.DatabaseError, ps.ProgrammingError) as error: print(error)

self.conn.rollback() return False

def count(self, table\_name: str):

return self.get\_el(f"select count(\*) from public.\"{table\_name}\"")

def find(self, table\_name: str, key\_name: str, key\_value: int):

return self.get\_el(f"select count(\*) from public.\"{table\_name}\" where

{key\_name}={key\_value}")

def max(self, table\_name: str, key\_name: str):

return self.get\_el(f"select max({key\_name}) from public.\"{table\_name}\"")

def min(self, table\_name: str, key\_name: str):

return self.get\_el(f"select min({key\_name}) from public.\"{table\_name}\"")

def print\_products(self) -> None:

table = self.get(f"SELECT \* FROM public.\"Product\"") print('Product table:')

for row in table:

print('id\_product:', row[0], '\ttitle:', row[1], '\tprice:', row[2], '\ tcategory:', row[3],

'\tid\_catalog:', row[4], '\tid\_order:', row[5]) print(' ')

def print\_order(self) -> None:

table = self.get(f"SELECT \* FROM public.\"Order\"") print('Order table:')

for row in table:

print('id\_order:', row[0], '\tcustomer\_name:', row[1], '\tid\_shop:', row[2], '\tdate:', row[3])

print(' ')

def print\_catalog(self) -> None:

table = self.get(f"SELECT \* FROM public.\"Catalog\"") print('Catalog table:')

for row in table:

print('id\_catalog:', row[0], '\tname:', row[1], '\tid\_shop:', row[2], '\ tpid\_catalog:', row[3])

print(' ')

def print\_shop(self) -> None:

table = self.get(f"SELECT \* FROM public.\"Shop\"") print('Shop table:')

for row in table:

print('id\_shop:', row[0], '\taddress:', row[1], '\tname:', row[2]) print(' ')

def delete\_data(self, table\_name: str, key\_name: str, key\_value) -> None: self.request(f"DELETE FROM public.\"{table\_name}\" WHERE

{key\_name}={key\_value};")

def update\_data\_product(self, key\_value: int, title: str, price: float, category: str, id\_catalog: int, id\_order: int) -> None:

self.request(f"UPDATE public.\"Product\" SET title=\'{title}\', price={price}, category=\'{category}\', "

f"id\_catalog={id\_catalog}, id\_order={id\_order} WHERE id\_product={key\_value};")

self.request(f"UPDATE public.\"Product\" SET title=\'{title}\', price={price}, category=\'{category}\', "

f"id\_catalog={id\_catalog}, id\_order={id\_order} WHERE id\_product={key\_value};")

def update\_data\_order(self, key\_value: int, customer\_name: str, id\_shop: int, date: datetime.datetime) -> None:

self.request(f"UPDATE public.\"Order\" SET customer\_name=\'{customer\_name}\', id\_shop={id\_shop}, "

f"date=\'{date}\' WHERE id\_order={key\_value};")

def update\_data\_catalog(self, key\_value: int, name: str, id\_shop: int, pid\_catalog: int) -> None:

self.request(f"UPDATE public.\"Catalog\" SET name=\'{name}\', id\_shop={id\_shop}, "

f"pid\_catalog={pid\_catalog} WHERE id\_order={key\_value};")

def update\_data\_shop(self, key\_value: int, address: str, name: str) -> None: self.request(f"UPDATE public.\"Shop\" SET address=\'{address}\', "

f"name=\'{name}\' WHERE id\_shop={key\_value};")

def insert\_data\_product(self, id\_product: int, title: str, price: float, category: str, id\_catalog: int,

id\_order: int) -> None:

self.request(f"insert into public.\"Product\" (id\_product, title, price, category, id\_catalog, id\_order) "

f"VALUES ({id\_product}, \'{title}\', {price}, \'{category}\',

{id\_catalog}, {id\_order});")

def insert\_data\_order(self, id\_order: int, customer\_name: str, id\_shop: int, date: datetime.datetime) -> None:

self.request(f"insert into public.\"Order\" (id\_order, customer\_name, id\_shop,

date) "

\'{date}\');")

f"VALUES ({id\_order}, \'{customer\_name}\', {id\_shop},

def insert\_data\_catalog(self, id\_catalog: int, name: str, id\_shop: int, pid\_catalog: int) -> None:

self.request(f"insert into public.\"Catalog\" (id\_catalog, name, id\_shop, pid\_catalog) "

f"VALUES ({id\_catalog}, \'{name}\', {id\_shop}, {pid\_catalog});")

def insert\_data\_shop(self, id\_shop: int, address: str, name: str) -> None: self.request(f"insert into public.\"Shop\" (id\_shop, address, name) "

f"VALUES ({id\_shop}, \'{address}\', \'{name}\');")

def product\_data\_generator(self, times: int) -> None: for i in range(times):

self.request("insert into public.\"Product\""

"select (SELECT MAX(id\_product)+1 FROM public.\"Product\"), " "array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)):: integer) \

FROM generate\_series(1, FLOOR(RANDOM()\*(10-4)+4):: integer)), ''), "

"FLOOR(RANDOM()\*(100000-1)+1),"

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) \

FROM generate\_series(1, FLOOR(RANDOM()\*(10-4)+4):: integer)), \ ''), "

"(SELECT id\_catalog FROM public.\"Catalog\" LIMIT 1 OFFSET (round(random() \* "

"((SELECT COUNT(id\_catalog) FROM public.\"Catalog\")-1))))," "(SELECT id\_order FROM public.\"Order\" LIMIT 1 OFFSET " "(round(random() \* ((SELECT COUNT(id\_order) FROM

public.\"Order\")-1))));")

def order\_data\_generator(self, times: int) -> None: for i in range(times):

self.request("insert into public.\"Order\" select (SELECT (MAX(id\_order)

+1) FROM public.\"Order\"), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25))

:: integer) " integer)), ''), "

"FROM generate\_series(1, FLOOR(RANDOM()\*(10-3)+3)::

"(SELECT id\_shop FROM public.\"Shop\" LIMIT 1 OFFSET " "(round(random() \*((SELECT COUNT(id\_shop) FROM

public.\"Shop\")-1)))), "

"(SELECT to\_timestamp(1549634400+random()\*70071999));")

def catalog\_data\_generator(self, times: int) -> None: for i in range(times):

self.request("insert into public.\"Catalog\" select (SELECT MAX(id\_catalog)+1 FROM public.\"Catalog\"), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25))

:: integer) " integer)), ''), "

"FROM generate\_series(1, FLOOR(RANDOM()\*(15-5)+5)::

"(SELECT id\_shop FROM public.\"Shop\" LIMIT 1 OFFSET " "(round(random() \*((SELECT COUNT(id\_shop) FROM

public.\"Shop\")-1)))), "

"(SELECT id\_catalog FROM public.\"Catalog\" LIMIT 1 OFFSET " "(round(random() \* ((SELECT COUNT(id\_catalog) FROM

public.\"Catalog\")-1))));")

def shop\_data\_generator(self, times: int) -> None: for i in range(times):

self.request("insert into public.\"Shop\" select (SELECT MAX(id\_shop)+1 FROM public.\"Shop\"), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "

"FROM generate\_series(1, FLOOR(RANDOM()\*(25-10)+10):: integer)), ''), "

"array\_to\_string(ARRAY(SELECT chr((97 + round(random() \* 25)) :: integer) "

"FROM generate\_series(1, FLOOR(RANDOM()\*(10-4)+4):: integer)), ''); ")

def search\_data\_two\_tables(self, table1\_name: str, table2\_name: str, table1\_key, table2\_key, search: str):

found = self.get(f"select \* from public.\"{table1\_name}\" as one inner join public.\"{table2\_name}\" as two "

f"on one.\"{table1\_key}\"=two.\"{table2\_key}\" " f"where {search}")

print('search result:') for row in found:

for i in range(0, len(row)): print(row[i])

print(' ')

def search\_data\_three\_tables(self, table1\_name: str, table2\_name: str, table3\_name: str,

table1\_key, table2\_key, table3\_key, table13\_key, search: str):

found = self.get(f"select \* from public.\"{table1\_name}\" as one inner join public.\"{table2\_name}\" as two "

f"on one.\"{table1\_key}\"=two.\"{table2\_key}\" inner join public.\"{table3\_name}\" as three "

f"on three.\"{table3\_key}\"=one.\"{table13\_key}\"" f"where {search}")

print('search result:') for row in found:

for i in range(0, len(row)): print(row[i])

print(' ')

def search\_data\_all\_tables(self, table1\_name: str, table2\_name: str, table3\_name: str, table4\_name: str,

table1\_key, table2\_key, table3\_key, table13\_key, table4\_key, table24\_key,

search: str):

found = self.get(f"select \* from public.\"{table1\_name}\" as one inner join public.\"{table2\_name}\" as two "

f"on one.\"{table1\_key}\"=two.\"{table2\_key}\" inner join public.\"{table3\_name}\" as three "

f"on three.\"{table3\_key}\"=one.\"{table13\_key}\" inner join public.\"{table4\_name}\" as four "

f"on four.\"{table4\_key}\"=two.\"{table24\_key}\"" f"where {search}")

print('search result:') for row in found:

for i in range(0, len(row)): print(row[i])

print(' ')

def numeric\_search(self, a: int, b: int, key: str): return f"{a}<{key} and {key}<{b}"

def string\_search(self, string: str, key: str): return f"{key} LIKE \'{string}\'"

def date\_search(self, datetime1: datetime.datetime, datetime2: datetime.datetime, key: str):

return f"{key} BETWEEN \'{datetime1}\' AND \'{datetime2}\'"