**Національний технічний університет України**

**“Київський політехнічний інститут”**

**Факультет прикладної математики**

**Кафедра системного програмування і спеціалізованих комп’ютерних систем**

**РОЗРАХУНКОВО-ГРАФІЧНА РОБОТА**

***з дисципліни***

***“*** **"Бази даних та засоби управління*”***

# ТЕМА: “ Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL"

Виконав студент групи: КВ-22

ПІБ: Марчук Дмитро Андрійович

Перевірив: Павловський В. І.

**Київ – 2024**

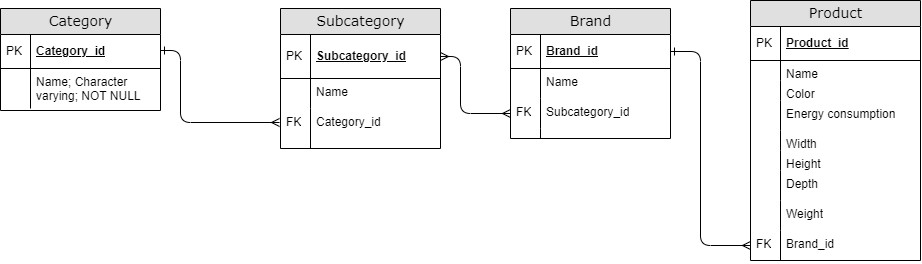
**Мета:** здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

**Завдання:**

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-поданняконтролер).

Використані бібліотеки: psycopg2 (для зв’язку з СУБД) , time (для вимірів часу)

# Відомості про обрану предметну галузь з лабораторної роботи №1

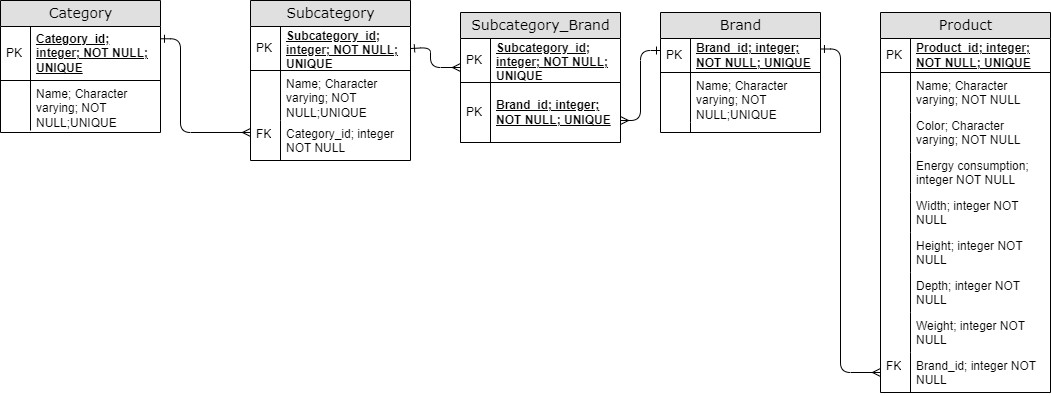


ER-діагарама побудована за нотацією «Crow`s foot».

# Опис предметної галузі

Дана предметна галузь реалізує електронний довідник для зберігання технічних характеристик товарів.

# Перетворення моделі у схему бази даних



# Перелік сутностей з описом їх призначення

Предметна галузь включає в себе 4 сутності , в кожної сутності є атрибут/ти:

1. Category (Category\_id (PK) , Name)
2. Subcategory (Subcategory\_id (PK) , Name , Category\_id (FK))
3. Brand (Brand\_id (PK) , Name , Subcategory\_id (FK))
4. Product (Product\_id (PK) , Name , color, Width, Height, Deepth, Energy Consumption , Weight, Brand\_id (FK))

Сутність Category описує категорії товарів. Наприклад: «Ноутбуки та комп’ютери». Має атрибут «Ім’я».

Сутність Subcategory описує підкатегорію категорії. Наприклад:

Підкатегорія «Монітори» в категорії : «Ноутбуки та комп’ютери». Має атрибути «Ім’я».

Сутність Brand описує бренд/фірму/виробника певного товару. Наприклад бренд «msi». Має атрибути «Ім’я».

Сутність Product описує сам товар. Має атрибути: Ім’я, колір, енергоспоживання, габарити, вага.

# Зв’язки між сутностями

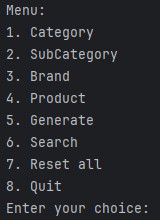
Зв’язок між Category і Subcategory: В одній категорії може міститися багато підкатегорій, але одна підкатегорія може міститися тільки в одній категорії.

1:N

Зв’язок між Subcategory і Brand: В одній підкатегорії може знаходитися багато різних брендів , але і один і той же бренд може знаходитися в різних підкатегоріях. N:M

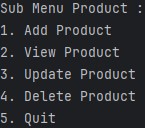
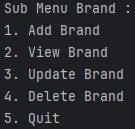
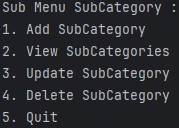
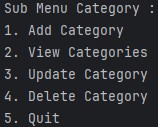
Зв’язок між Brand і Product: Бренд може випускати багато продуктів , але у одного продукта може бути тільки один виробник. 1:N.

# Схема меню



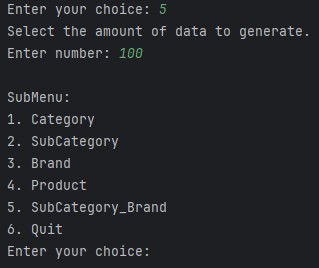
На скріншоті показане головне меню програми. Вони містить 8 пунктів: Пункти 1-4 призначені для переходу до роботи з відповідними таблицями, пункт 5 дозволяє генерувати дані для пізніше обраної таблиці, пункт 6 дозволяє зробити 1 з 3 пошукових запитів для БД, пункт 7 очищує всі таблиці, пункт 8 – вихід з програми.

Також для всіх таблиць (окрім додаткової) реалізовано окреме підменю яке викликається після обрання відповідної таблиці у головному меню.

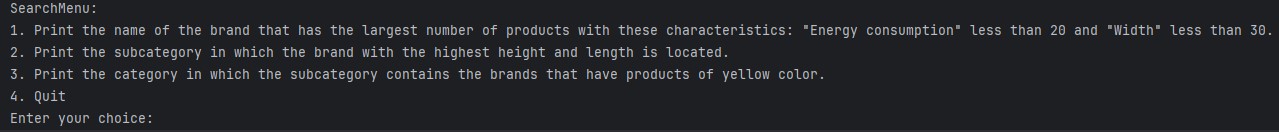


У цих підменю доступні операції: додавання, перегляду, редагування та вилученя даних у таблицях бази даних.

Для пункту 5(Генерування даних) також передбачене підменю в якому можна обрати таблицю для якої , ви хочете згенерувати дані, попередньо обравши кількість даних дял генерації.



Для пункту 6(Пошук) було створено 3 запити , які реалізують пошук у 2-х,3-х та 4-х таблицях. Для цього є відповідне підменю з запитами.

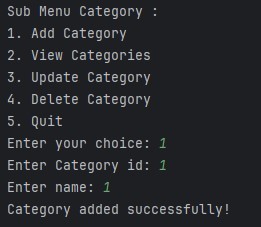


# Програмне забезпечення

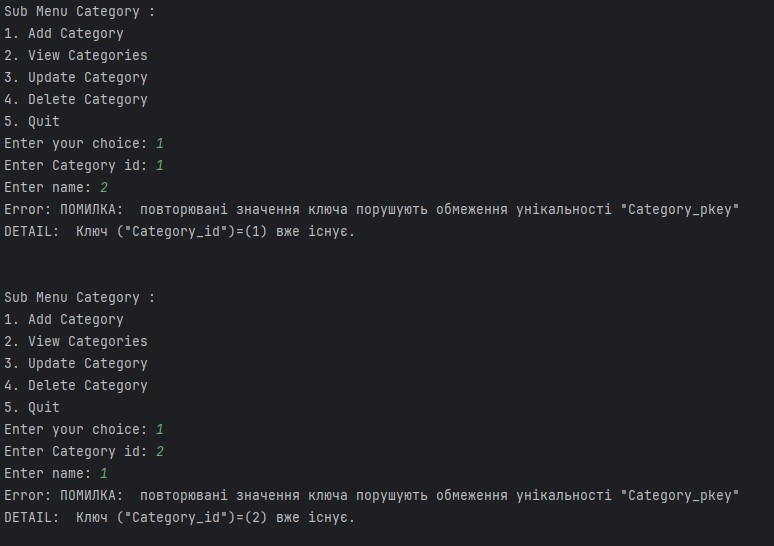
Програма була написана на мові програмуванні Python з використанням бібліотеки Psycopg2.

# Завдання №1

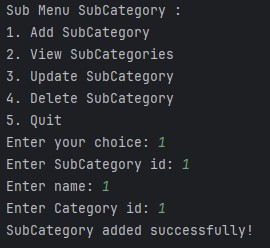
Зробимо додавання даних у таблицю Category:



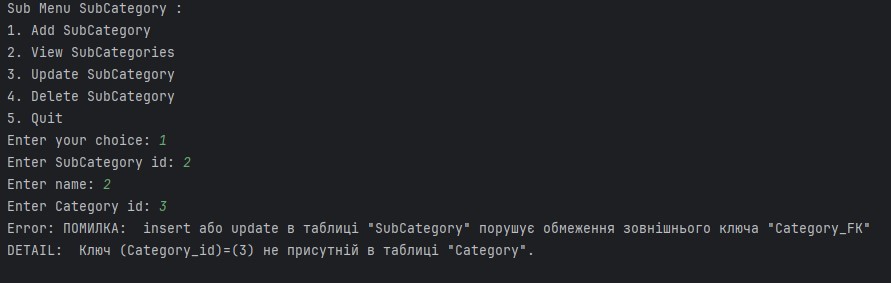
Можна побачити , що програма фіксує дублювання даних та не дає додати їх:



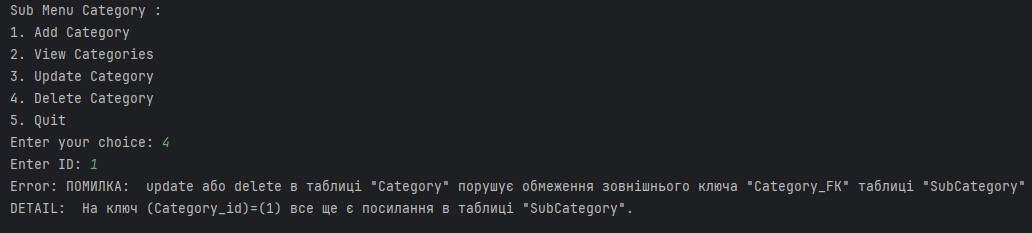
Додамо дані в таблицю Subcategory:



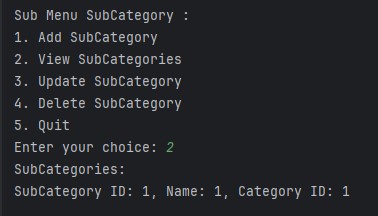
При спробі вставки в дочірню таблицю Subcategory неіснуючого значення Category\_id видається відповідна помилка:



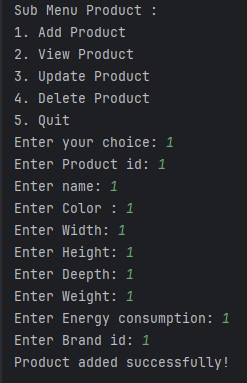
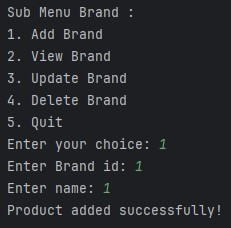
При спробі вилучення даних з батьківської таблиці без попередньго видалення даних з дочірньої виникає відповідна помилка:

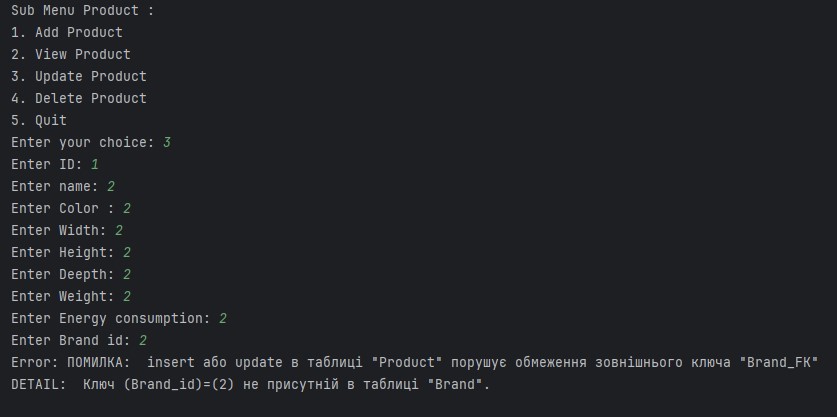


Видалення не було здійснено:

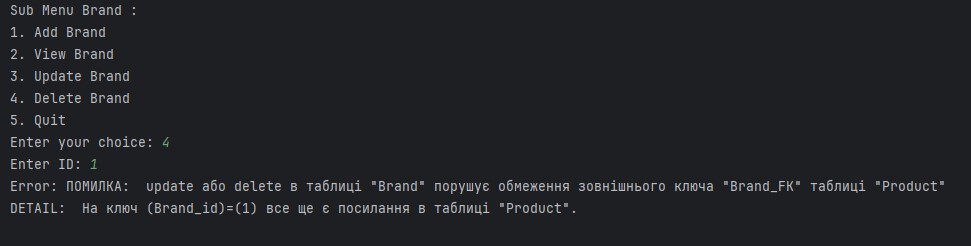
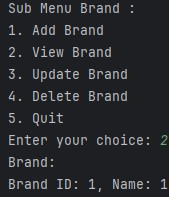
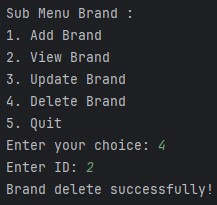
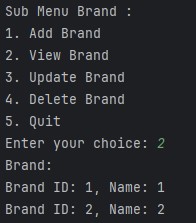


Так само і з оновленням даних:



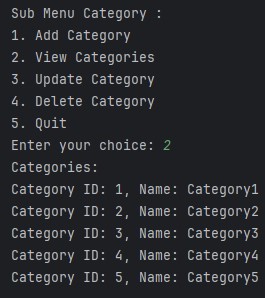
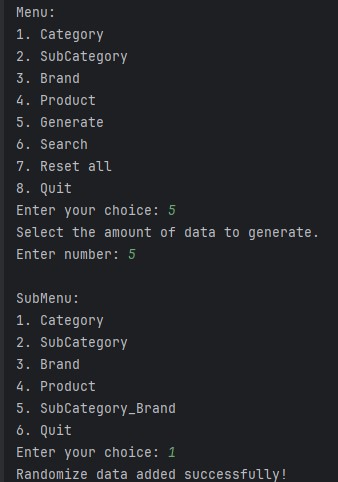


«Неприв’язані» дані з батьківської таблиці можна успішно видаляти:

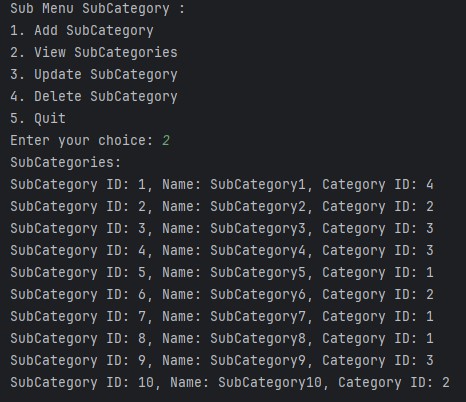
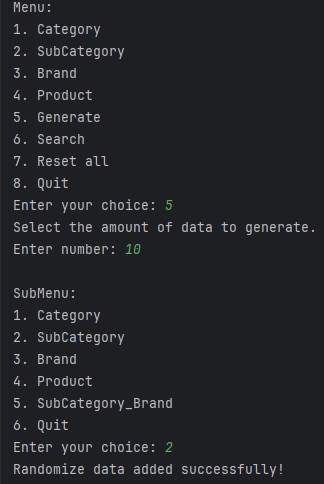


# Завдання №2

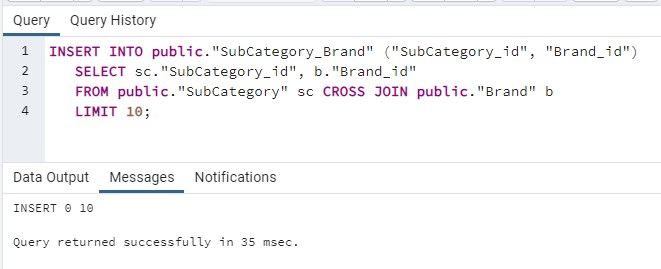
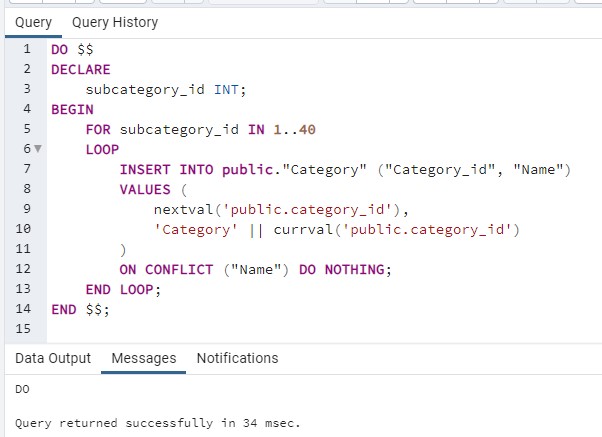
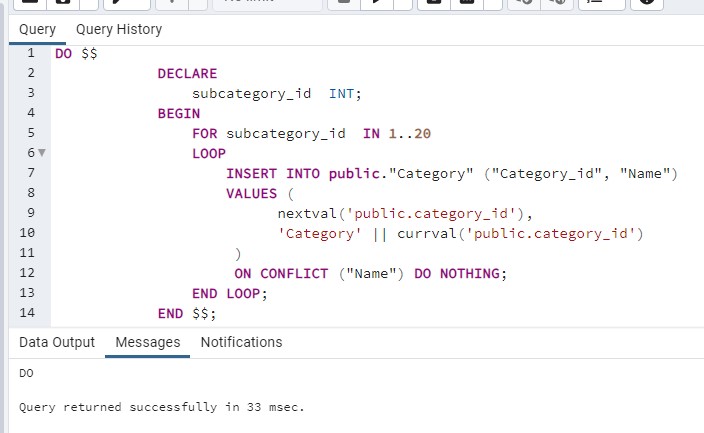
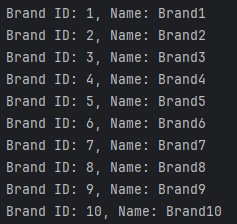
Генерація 5 рандомних значень для таблиці Category:



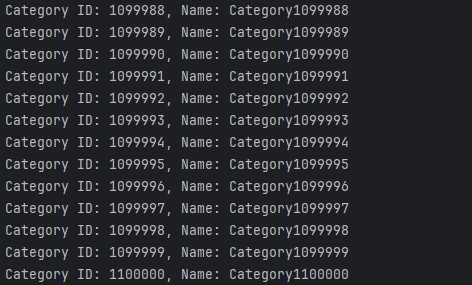
Аналогічна операція для таблиці Subcategory:



Можна побачити що значення зовнішнього ключа Category\_id обирається з існуючих значень з таблиці Category. Тобто обмеження зовнішнього ключа не порушується.

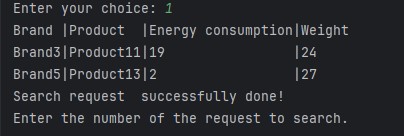
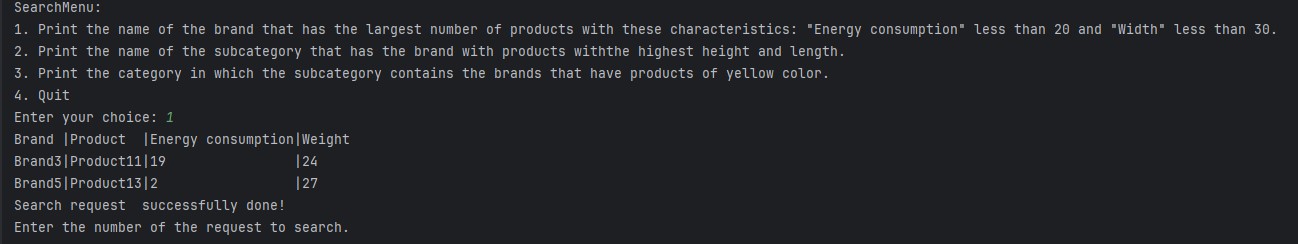


Приклад генерації великої кількості даних:

 є

# Завдання №3

Перший запит виводить назву бренду, який має найбільшу кількість товарів з такими характеристиками: "Енергоспоживання" менше 20 та "Ширина" менше 30.



Час виконання запиту на 10000 елементів таблиці Product і 5000 елементів таблиці Brand:



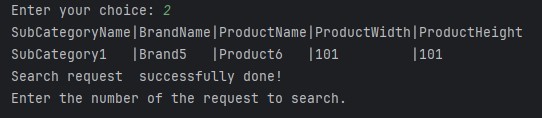
Час виконання запиту на 10000 елементів таблиці Product, 5000 елементів таблиці Brand і 5000 елементів таблиці SubCategory:



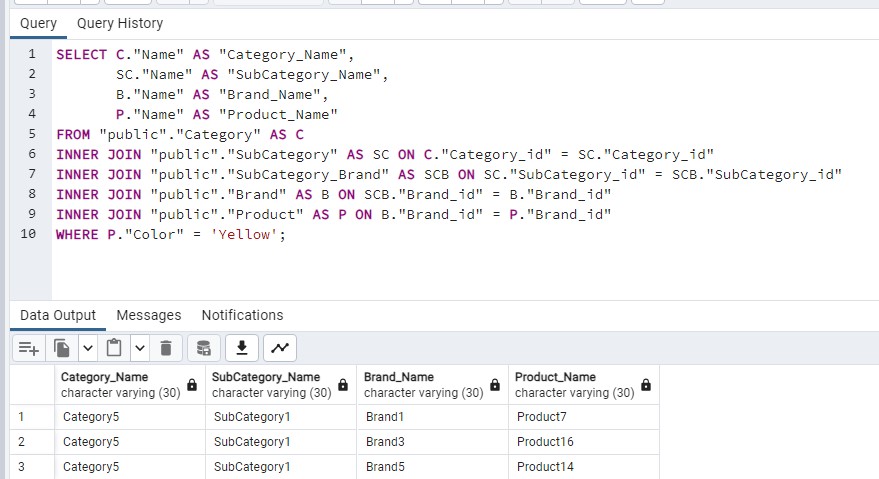
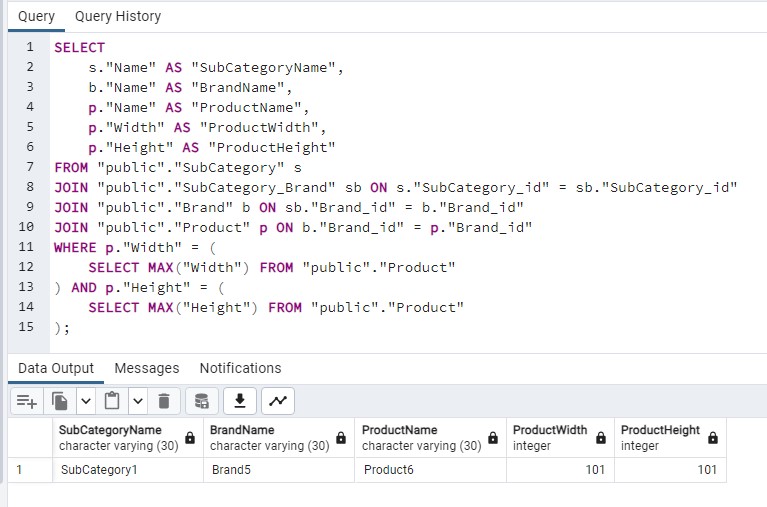
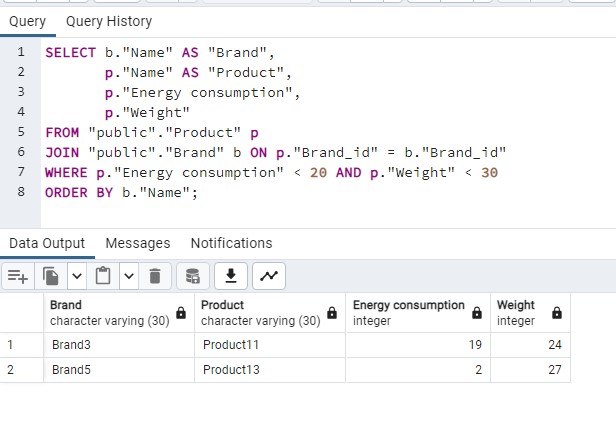
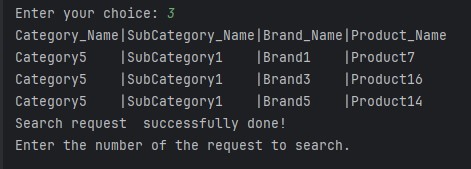
Час виконання запиту на 10000 елементів таблиці Product, 5000 елементів таблиці Brand, 5000 елементів таблиці SubCategory і 1000 елементів таблиці Category:



Другий запит назву підкатегорії, в якій є бренд з товарами з найбільшою висотою та довжиною.



Третій запит виводить категорію, підкатегорія якої містить бренди, що мають товари жовтого кольору.



# Завдання №4

**import** psycopg2

**class** Model: **def** \_\_init\_\_(self): self.conn = psycopg2.connect( dbname='RGR', user='postgres', password='1111', host='localhost', port=5432

)

self.create\_table\_Category() self.create\_table\_SubCategory() self.create\_table\_Brand() self.create\_table\_Product() self.create\_table\_SubCategory\_Brand()

**def** create\_table\_Category(self):

c = self.conn.cursor()

c.execute('''

CREATE TABLE IF NOT EXISTS "Category" (

"Category\_id" integer NOT NULL,

"Name" character varying(30) NOT NULL,

CONSTRAINT "Category\_pkey" PRIMARY KEY ("Category\_id"),

CONSTRAINT name\_un UNIQUE ("Name")

)

''')

# Check if the table exists

c.execute("SELECT EXISTS (SELECT 1 FROM information\_schema.tables WHERE table\_name = 'Category')")

table\_exists = c.fetchone()[0]

**if** **not** table\_exists:

# Table does not exist, so create it

c.execute('''

CREATE TABLE "Category" (

"Category\_id" integer NOT NULL,

"Name" character varying(30) NOT NULL,

CONSTRAINT "Category\_pkey" PRIMARY KEY ("Category\_id"),

CONSTRAINT name\_un UNIQUE ("Name")

)

''')

self.conn.commit()

**def** create\_table\_SubCategory(self):

c = self.conn.cursor()

c.execute('''

CREATE TABLE IF NOT EXISTS "SubCategory" (

"SubCategory\_id" integer NOT NULL,

"Name" character varying(30) NOT NULL,

"Category\_id" integer NOT NULL,

CONSTRAINT "SubCategory\_pkey" PRIMARY KEY

("SubCategory\_id"),

CONSTRAINT name\_un2 UNIQUE ("Name"),

CONSTRAINT "Category\_FK" FOREIGN KEY ("Category\_id")

REFERENCES public."Category" ("Category\_id") MATCH SIMPLE

)

''')

# Check if the table exists

c.execute("SELECT EXISTS (SELECT 1 FROM information\_schema.tables WHERE table\_name = 'SubCategory')")

table\_exists = c.fetchone()[0]

**if** **not** table\_exists:

# Table does not exist, so create it

c.execute('''

CREATE TABLE "SubCategory" (

"SubCategory\_id" integer NOT NULL,

"Name" character varying(30) NOT NULL,

"Category\_id" integer NOT NULL,

CONSTRAINT "SubCategory\_pkey" PRIMARY KEY

("SubCategory\_id"),

CONSTRAINT name\_un2 UNIQUE ("Name"),

CONSTRAINT "Category\_FK" FOREIGN KEY ("Category\_id")

REFERENCES public."Category" ("Category\_id") MATCH SIMPLE

)

''')

self.conn.commit()

**def** create\_table\_Brand(self): c = self.conn.cursor()

c.execute('''

CREATE TABLE IF NOT EXISTS "Brand" (

"Brand\_id" integer NOT NULL,

"Name" character varying(30) NOT NULL,

CONSTRAINT "Brand\_pkey" PRIMARY KEY ("Brand\_id"),

CONSTRAINT name\_un3 UNIQUE ("Name")

)

''')

# Check if the table exists

c.execute("SELECT EXISTS (SELECT 1 FROM information\_schema.tables WHERE table\_name = 'Brand')")

table\_exists = c.fetchone()[0]

**if** **not** table\_exists:

# Table does not exist, so create it

c.execute('''

CREATE TABLE "Brand" (

"Brand\_id" integer NOT NULL,

"Name" character varying(30) NOT NULL,

CONSTRAINT "Brand\_pkey" PRIMARY KEY ("Brand\_id"), CONSTRAINT name\_un3 UNIQUE ("Name")

)

''')

self.conn.commit()

**def** create\_table\_Product(self):

c = self.conn.cursor()

c.execute('''

CREATE TABLE IF NOT EXISTS "Product" (

"Product\_id" integer NOT NULL,

"Name" character varying(30) NOT NULL,

"Color" character varying(30) NOT NULL,

"Width" integer NOT NULL,

"Height" integer NOT NULL,

"Deepth" integer NOT NULL,

"Weight" integer NOT NULL,

"Energy consumption" integer NOT NULL,

"Brand\_id" integer NOT NULL,

CONSTRAINT "Product\_pkey" PRIMARY KEY ("Product\_id"),

CONSTRAINT "Brand\_FK" FOREIGN KEY ("Brand\_id")

REFERENCES public."Brand" ("Brand\_id") MATCH SIMPLE

)

''')

# Check if the table exists

c.execute("SELECT EXISTS (SELECT 1 FROM information\_schema.tables WHERE table\_name = 'Product')")

table\_exists = c.fetchone()[0]

**if** **not** table\_exists:

# Table does not exist, so create it

c.execute('''

CREATE TABLE "Product" (

"Product\_id" integer NOT NULL,

"Name" character varying(30) NOT NULL,

"Color" character varying(30) NOT NULL,

"Width" integer NOT NULL,

"Height" integer NOT NULL,

"Deepth" integer NOT NULL,

"Weight" integer NOT NULL,

"Energy consumption" integer NOT NULL,

"Brand\_id" integer NOT NULL,

CONSTRAINT "Product\_pkey" PRIMARY KEY ("Product\_id"),

CONSTRAINT "Brand\_FK" FOREIGN KEY ("Brand\_id")

REFERENCES public."Brand" ("Brand\_id") MATCH SIMPLE

)

''')

self.conn.commit()

**def** create\_table\_SubCategory\_Brand(self):

c = self.conn.cursor()

c.execute('''

CREATE TABLE IF NOT EXISTS "SubCategory\_Brand" (

"SubCategory\_id" integer NOT NULL,

"Brand\_id" integer NOT NULL,

CONSTRAINT "SubCategory\_Brand\_pkey" PRIMARY KEY

("SubCategory\_id", "Brand\_id"),

CONSTRAINT "Brand\_FK" FOREIGN KEY ("Brand\_id")

REFERENCES public."Brand" ("Brand\_id") MATCH SIMPLE,

CONSTRAINT "SubCategory\_FK" FOREIGN KEY

("SubCategory\_id") REFERENCES public."SubCategory"

("SubCategory\_id") MATCH SIMPLE

)

''')

# Check if the table exists

c.execute("SELECT EXISTS (SELECT 1 FROM

information\_schema.tables WHERE table\_name = 'SubCategory\_Brand')")

table\_exists = c.fetchone()[0]

**if** **not** table\_exists:

# Table does not exist, so create it

c.execute('''

CREATE TABLE "SubCategory\_Brand" (

"SubCategory\_id" integer NOT NULL,

"Brand\_id" integer NOT NULL,

CONSTRAINT "SubCategory\_Brand\_pkey" PRIMARY KEY

("SubCategory\_id", "Brand\_id"),

CONSTRAINT "Brand\_FK" FOREIGN KEY ("Brand\_id")

REFERENCES public."Brand" ("Brand\_id") MATCH SIMPLE,

CONSTRAINT "SubCategory\_FK" FOREIGN KEY

("SubCategory\_id") REFERENCES public."SubCategory"

("SubCategory\_id") MATCH SIMPLE

)

''')

self.conn.commit() **def** add\_Category(self, Category\_id, Name): c = self.conn.cursor() **try**:

c.execute('INSERT INTO "Category" ("Category\_id", "Name") VALUES (%s, %s)', (Category\_id, Name))

self.conn.commit()

**return** "Category added successfully!" **except** psycopg2.IntegrityError **as** e: self.conn.rollback() error\_message = str(e) **return** "Error: " + error\_message **finally**:

c.close()

**def** get\_all\_Category(self): c = self.conn.cursor()

c.execute('SELECT \* FROM "Category"')

**return** c.fetchall()

**def** update\_Category(self, Category\_id, Name): c = self.conn.cursor() **try**:

c.execute('UPDATE "Category" SET "Name"=%s WHERE "Category\_id"=%s', (Name, Category\_id)) self.conn.commit()

**return** "Category update successfully!" **except** psycopg2.IntegrityError **as** e: self.conn.rollback() error\_message = str(e) **return** "Error: " + error\_message **finally**:

c.close() **def** delete\_Category(self, Category\_id): c = self.conn.cursor() **try**:

c.execute('DELETE FROM "Category" WHERE "Category\_id"=%s', (Category\_id,)) self.conn.commit()

**return** "Category delete successfully!" **except** psycopg2.IntegrityError **as** e: self.conn.rollback() error\_message = str(e) **return** "Error: " + error\_message **finally**:

c.close() **def** add\_SubCategory(self, SubCategory\_id, Name,Category\_id): c = self.conn.cursor() **try**:

c.execute('INSERT INTO "SubCategory" ("SubCategory\_id",

"Name","Category\_id") VALUES (%s, %s,%s)', ( SubCategory\_id, Name,Category\_id)) self.conn.commit()

**return** "SubCategory added successfully!" **except** psycopg2.IntegrityError **as** e: self.conn.rollback() error\_message = str(e) **return** "Error: " + error\_message **finally**:

c.close()

**def** get\_all\_SubCategory(self): c = self.conn.cursor()

c.execute('SELECT \* FROM "SubCategory"')

**return** c.fetchall()

**def** update\_SubCategory(self, SubCategory\_id,

Name,Category\_id): c = self.conn.cursor() **try**:

c.execute('UPDATE "SubCategory" SET "Name"=%s, "Category\_id"=%s WHERE "SubCategory\_id"=%s', (Name, Category\_id, SubCategory\_id)) self.conn.commit()

**return** "SubCategory update successfully!" **except** psycopg2.IntegrityError **as** e: self.conn.rollback() error\_message = str(e) **return** "Error: " + error\_message **finally**:

c.close() **def** delete\_SubCategory(self, SubCategory\_id): c = self.conn.cursor() **try**:

c.execute('DELETE FROM "SubCategory" WHERE "SubCategory\_id"=%s', (SubCategory\_id,)) self.conn.commit()

**return** "SubCategory delete successfully!" **except** psycopg2.IntegrityError **as** e: self.conn.rollback() error\_message = str(e) **return** "Error: " + error\_message **finally**:

c.close() **def** add\_Brand(self, Brand\_id, Name): c = self.conn.cursor() **try**:

c.execute('INSERT INTO "Brand" ("Brand\_id", "Name") VALUES (%s, %s)', (Brand\_id, Name)) self.conn.commit()

**return** "Product added successfully!" **except** psycopg2.IntegrityError **as** e: self.conn.rollback() error\_message = str(e) **return** "Error: " + error\_message **finally**:

c.close() **def** get\_all\_Brand(self): c = self.conn.cursor()

c.execute('SELECT \* FROM "Brand"')

**return** c.fetchall()

**def** update\_Brand(self, Brand\_id, Name): c = self.conn.cursor() **try**:

c.execute('UPDATE "Brand" SET "Name"=%s WHERE "Brand\_id"=%s', (Name, Brand\_id)) self.conn.commit()

**return** "Brand update successfully!" **except** psycopg2.IntegrityError **as** e: self.conn.rollback() error\_message = str(e) **return** "Error: " + error\_message **finally**:

c.close() **def** delete\_Brand(self, Brand\_id):

c = self.conn.cursor() **try**:

c.execute('DELETE FROM "Brand" WHERE "Brand\_id"=%s', (Brand\_id,))

self.conn.commit()

**return** "Brand delete successfully!" **except** psycopg2.IntegrityError **as** e: self.conn.rollback() error\_message = str(e) **return** "Error: " + error\_message **finally**:

c.close()

**def** add\_Product(self, Product\_id, Name, Color, Width, Height,

Deepth, Weight, Energy\_consumption, Brand\_id):

c = self.conn.cursor() **try**:

c.execute('INSERT INTO "Product" ("Product\_id", "Name", "Color", "Width", "Height", "Deepth", "Weight", "Energy

consumption", "Brand\_id") VALUES (%s, %s,%s, %s,%s, %s,%s, %s,%s)',

(Product\_id, Name, Color, Width, Height, Deepth, Weight, Energy\_consumption, Brand\_id)) self.conn.commit()

**return** "Product added successfully!" **except** psycopg2.IntegrityError **as** e: self.conn.rollback() error\_message = str(e) **return** "Error: " + error\_message **finally**:

c.close() **def** get\_all\_Product(self): c = self.conn.cursor()

c.execute('SELECT \* FROM "Product"')

**return** c.fetchall()

**def** update\_Product(self, Product\_id, Name, Color, Width,

Height, Deepth, Weight, Energy\_consumption, Brand\_id): c = self.conn.cursor() **try**:

c.execute('UPDATE "Product" SET "Name"=%s, "Color"=%s, "Width"=%s, "Height"=%s, "Deepth"=%s, "Weight"=%s, "Energy consumption"=%s, "Brand\_id"=%s WHERE "Product\_id"=%s', ( Name,

Color, Width, Height, Deepth, Weight, Energy\_consumption,

Brand\_id,Product\_id)) self.conn.commit()

**return** "Product update successfully!" **except** psycopg2.IntegrityError **as** e: self.conn.rollback() error\_message = str(e) **return** "Error: " + error\_message **finally**:

c.close() **def** delete\_Product(self, Product\_id): c = self.conn.cursor() **try**:

c.execute('DELETE FROM "Product" WHERE "Product\_id"=%s', (Product\_id,)) self.conn.commit()

**return** "Product delete successfully!" **except** psycopg2.IntegrityError **as** e: self.conn.rollback() error\_message = str(e) **return** "Error: " + error\_message **finally**:

c.close() **def** generate\_category(self, number):

c = self.conn.cursor()

c.execute( '''

DO $$ DECLARE

subcategory\_id INT;

BEGIN

FOR subcategory\_id IN 1..%s

LOOP

INSERT INTO public."Category" ("Category\_id",

"Name")

VALUES (

nextval('public.category\_id'),

'Category' || currval('public.category\_id')

)

ON CONFLICT ("Name") DO NOTHING;

END LOOP;

END $$;

''',

(number,)

)

self.conn.commit()

**def** generate\_subcategory(self, number):

c = self.conn.cursor()

c.execute( '''

DO $$ DECLARE

subcategory\_id INT;

BEGIN

FOR subcategory\_id IN 1..%s

LOOP

INSERT INTO public."SubCategory"

("SubCategory\_id", "Name", "Category\_id") VALUES (

nextval('public.subcategory\_id'),

'SubCategory' || currval('public.subcategory\_id'),

(SELECT "Category\_id" FROM public."Category" ORDER BY random() LIMIT 1)

);

END LOOP;

END $$;

''',

(number,)

)

self.conn.commit()

**def** generate\_brand(self, number):

c = self.conn.cursor()

c.execute( '''

DO $$ DECLARE

brand\_id INT;

BEGIN

FOR brand\_id IN 1..%s

LOOP

INSERT INTO public."Brand" ("Brand\_id", "Name") VALUES (

nextval('public.brand\_id'),

'Brand' || currval('public.brand\_id')

)

ON CONFLICT ("Name") DO NOTHING;

END LOOP;

END $$;

''',

(number,)

)

self.conn.commit()

**def** generate\_product(self, number):

c = self.conn.cursor()

c.execute( '''

DO $$ DECLARE

product\_id INT;

BEGIN

FOR product\_id IN 1..%s

LOOP

INSERT INTO public."Product" ("Product\_id", "Name", "Color", "Width", "Height", "Deepth", "Weight", "Energy consumption", "Brand\_id") VALUES (

nextval('public.product\_id'),

'Product' || currval('public.product\_id'),

CASE WHEN random() < 0.2 THEN 'Red'

WHEN random() < 0.4 THEN 'Blue'

WHEN random() < 0.6 THEN 'Green' WHEN random() < 0.8 THEN 'Yellow'

ELSE 'Purple' END,

(random() \* 100 + 1)::integer,

(random() \* 100 + 1)::integer,

(random() \* 100 + 1)::integer,

(random() \* 100 + 1)::integer,

(random() \* 100 + 1)::integer,

(SELECT "Brand\_id" FROM public."Brand"

ORDER BY random() LIMIT 1)

);

END LOOP;

END $$;

''',

(number,)

)

self.conn.commit()

**def** generate\_subcategory\_brand(self, number):

c = self.conn.cursor()

c.execute(

'INSERT INTO public."SubCategory\_Brand" ("SubCategory\_id", "Brand\_id") SELECT sc."SubCategory\_id",

b."Brand\_id" FROM public."SubCategory" sc CROSS JOIN public."Brand" b LIMIT %s;', (number,))

self.conn.commit()

**def** reset(self):

c = self.conn.cursor()

c.execute('DELETE FROM public."SubCategory\_Brand"; DELETE FROM public."Product"; DELETE FROM public."Brand"; DELETE FROM public."SubCategory"; DELETE FROM public."Category";SELECT setval(\'category\_id\', 0); SELECT setval(\'subcategory\_id\', 0);SELECT setval(\'brand\_id\', 0);SELECT setval(\'product\_id\',

0);')

self.conn.commit()

**def** search\_request1(self): c = self.conn.cursor()

c.execute('SELECT b."Name" AS "Brand", p."Name" AS

"Product", p."Energy consumption", p."Weight" FROM

"public"."Product" p JOIN "public"."Brand" b ON p."Brand\_id" =

b."Brand\_id" WHERE p."Energy consumption" < 20 AND p."Weight" < 30 ORDER BY b."Name";') self.conn.commit()

**return** c.fetchall(), c.description

**def** search\_request2(self): c = self.conn.cursor()

c.execute('SELECT s."Name" AS "SubCategoryName", b."Name"

AS "BrandName", p."Name" AS "ProductName",p."Width" AS

"ProductWidth",p."Height" AS "ProductHeight" FROM

"public"."SubCategory" s JOIN "public"."SubCategory\_Brand" sb ON

s."SubCategory\_id" = sb."SubCategory\_id" JOIN "public"."Brand" b ON sb."Brand\_id" = b."Brand\_id" JOIN "public"."Product" p ON

b."Brand\_id" = p."Brand\_id" WHERE p."Width" = (SELECT MAX("Width")

FROM "public"."Product") AND p."Height" = (SELECT MAX("Height")

FROM "public"."Product");') self.conn.commit()

**return** c.fetchall(), c.description

**def** search\_request3(self): c = self.conn.cursor()

c.execute('SELECT c."Name" AS "Category\_Name", sc."Name" AS

"SubCategory\_Name", b."Name" AS "Brand\_Name", p."Name" AS

"Product\_Name" FROM "public"."Category" AS c INNER JOIN

"public"."SubCategory" AS sc ON c."Category\_id" = sc."Category\_id"

INNER JOIN "public"."SubCategory\_Brand" AS scb ON

SC."SubCategory\_id" = scb."SubCategory\_id" INNER JOIN

"public"."Brand" AS b ON scb."Brand\_id" = b."Brand\_id" INNER JOIN "public"."Product" AS p ON b."Brand\_id" = p."Brand\_id" WHERE

p."Color" = \'Yellow\';') self.conn.commit() **return** c.fetchall(), c.description

В даному модулі реалізуються всі запити до бази даних програми.

Є окремі методи для кожної таблиці.

Методи Create\_table\_.... – перевіряють чи існує відпрвідна таблиця , якщо її немає то він свторює її.

Методи add\_... – додають нові дані у відповідні таблиці.

Методи view\_... – показують дані у відповідих таблицях.

Методи update\_... – оновляють дані у відповідних таблицях.

Методи delete\_... - видаляють дані у відповідих таблицях.

Методи generate\_... - генерують дані для відповідних таблиць.

Метод reset – видаляє дані з усіх таблиць.

Методи search\_request\_... - роблять відповідні пошукові запити.

Посилання на репозиторій:

<https://github.com/Devded86/DB_KPI>