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"

Група: КВ-11

Виконала: Нестерук А.О.

Оцінка:

Київ – 2023

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

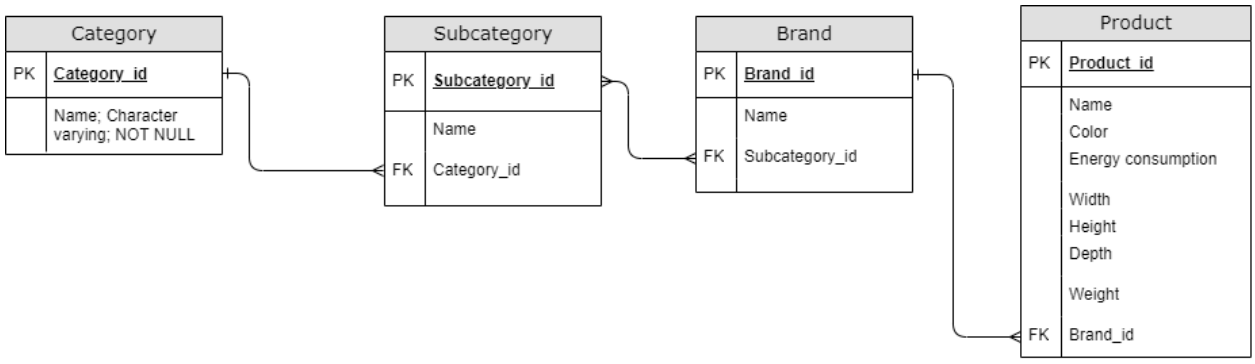
Посилання на телеграм та репозиторій:

https://t.me/jemapel_sasuke_uchiwa

https://github.com/FLeD-jk/RGR-BD

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

Відомості про обрану предметну галузь з лабораторної роботи №1

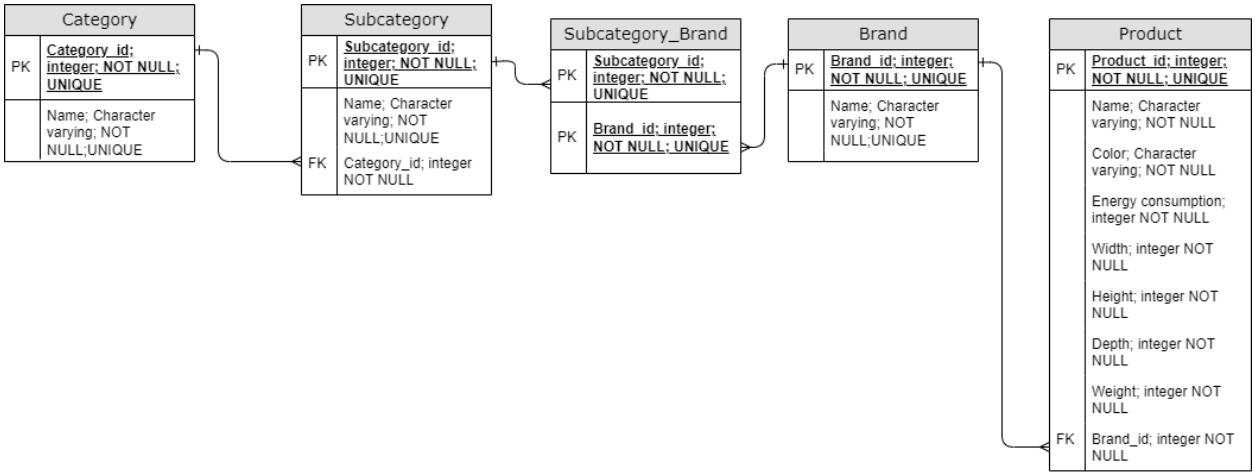


ER-діагарама побудована за нотацією «Crow`s foot».

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.

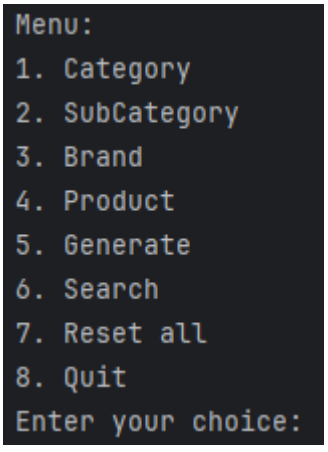
Зв’язки між сутностями

Зв’язок між Category і Subcategory: В одній категорії може міститися багато підкатегорій, але одна підкатегорія може міститися тільки в одній категорії. 1:N

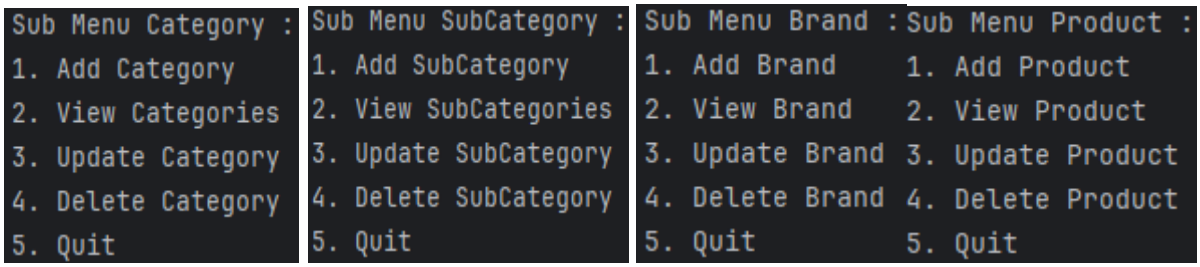
Зв’язок між Subcategory і Brand: В одній підкатегорії може знаходитися багато різних брендів , але і один і той же бренд може знаходитися в різних підкатегоріях. N:M

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

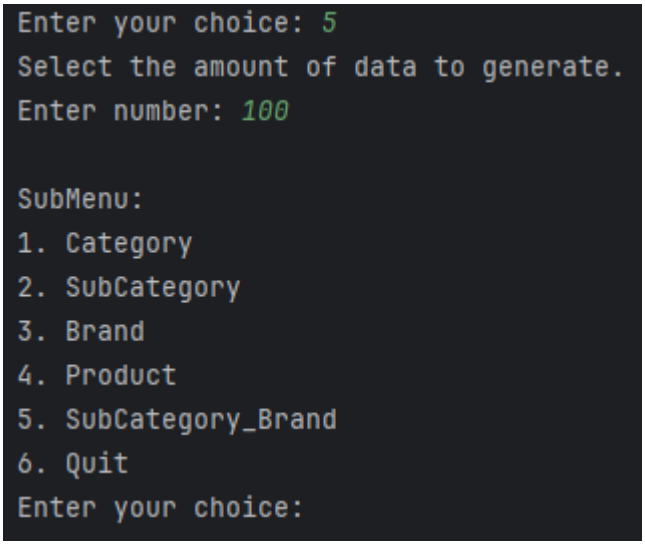
Схема меню



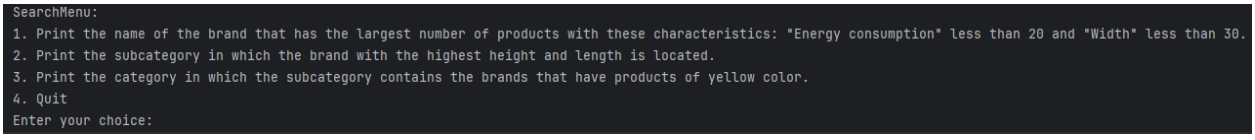
На скріншоті показане головне меню програми. Вони містить 8 пунктів: Пункти 1-4 призначені для переходу до роботи з відповідними таблицями, пункт 5 дозволяє генерувати дані для пізніше обраної таблиці, пункт 6 дозволяє зробити 1 з 3 пошукових запитів для БД, пункт 7 очищує всі таблиці, пункт 8 – вихід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

```
Sub Menu Category :
1. Add Category
2. View Categories
3. Update Category
4. Delete Category
5. Quit
Enter your choice: 1
Enter Category id: 1
Enter name: 1
Category added successfully!
```

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

```
Sub Menu Category :
1. Add Category
2. View Categories
3. Update Category
4. Delete Category
5. Quit
Enter your choice: 1
Enter Category id: 1
Enter name: 2
Error: ПОМИЛКА:  повторювані значення ключа порушують обмеження унікальності "Category_pkey"
DETAIL:  Ключ ("Category_id")=(1) вже існує.


Sub Menu Category :
1. Add Category
2. View Categories
3. Update Category
4. Delete Category
5. Quit
Enter your choice: 1
Enter Category id: 2
Enter name: 1
Error: ПОМИЛКА:  повторювані значення ключа порушують обмеження унікальності "Category_pkey"
DETAIL:  Ключ ("Category_id")=(2) вже існує.
```

Додамо дані в таблицю Subcategory:

```
Sub Menu SubCategory :
1. Add SubCategory
2. View SubCategories
3. Update SubCategory
4. Delete SubCategory
5. Quit
Enter your choice: 1
Enter SubCategory id: 1
Enter name: 1
Enter Category id: 1
SubCategory added successfully!
```

При спробі вставки в дочірню таблицю Subcategory неіснуючого значення Category_id видається відповідна помилка:

```
Sub Menu SubCategory :
1. Add SubCategory
2. View SubCategories
3. Update SubCategory
4. Delete SubCategory
5. Quit
Enter your choice: 1
Enter SubCategory id: 2
Enter name: 2
Enter Category id: 3
Error: ПОМИЛКА:  insert або update в таблиці "SubCategory" порушує обмеження зовнішнього ключа "Category_FK"
DETAIL:  Ключ (Category_id)=(3) не присутній в таблиці "Category".
```

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

```
Sub Menu Category :
1. Add Category
2. View Categories
3. Update Category
4. Delete Category
5. Quit
Enter your choice: 4
Enter ID: 1
Error: ПОМИЛКА:  update або delete в таблиці "Category" порушує обмеження зовнішнього ключа "Category_FK" таблиці "SubCategory"
DETAIL:  На ключ (Category_id)=(1) все ще є посилання в таблиці "SubCategory".
```

Видалення не було здійснено:

```
Sub Menu SubCategory :
1. Add SubCategory
2. View SubCategories
3. Update SubCategory
4. Delete SubCategory
5. Quit
Enter your choice: 2
SubCategories:
SubCategory ID: 1, Name: 1, Category ID: 1
```

Так само і з оновленням даних:

```
Sub Menu Brand :
1. Add Brand
2. View Brand
3. Update Brand
4. Delete Brand
5. Quit
Enter your choice: 1
Enter Brand id: 1
Enter name: 1
Product added successfully!
```

```
Sub Menu Product :
1. Add Product
2. View Product
3. Update Product
4. Delete Product
5. Quit
Enter your choice: 1
Enter Product id: 1
Enter name: 1
Enter Color : 1
Enter Width: 1
Enter Height: 1
Enter Deepth: 1
Enter Weight: 1
Enter Energy consumption: 1
Enter Brand id: 1
Product added successfully!
```

```
Sub Menu Product :
1. Add Product
2. View Product
3. Update Product
4. Delete Product
5. Quit
Enter your choice: 3
Enter ID: 1
Enter name: 2
Enter Color : 2
Enter Width: 2
Enter Height: 2
Enter Deepth: 2
Enter Weight: 2
Enter Energy consumption: 2
Enter Brand id: 2
Error: ПОМИЛКА:  insert або update в таблиці "Product" порушує обмеження зовнішнього ключа "Brand_FK"
DETAIL:  Ключ (Brand_id)=(2) не присутній в таблиці "Brand".
```

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

```
Sub Menu Brand :
1. Add Brand
2. View Brand
3. Update Brand
4. Delete Brand
5. Quit
Enter your choice: 2
Brand:
Brand ID: 1, Name: 1
Brand ID: 2, Name: 2
```

```
Sub Menu Brand :
1. Add Brand
2. View Brand
3. Update Brand
4. Delete Brand
5. Quit
Enter your choice: 4
Enter ID: 2
Brand delete successfully!
```

```
Sub Menu Brand :
1. Add Brand
2. View Brand
3. Update Brand
4. Delete Brand
5. Quit
Enter your choice: 2
Brand:
Brand ID: 1, Name: 1
```

```
Sub Menu Brand :
1. Add Brand
2. View Brand
3. Update Brand
4. Delete Brand
5. Quit
Enter your choice: 4
Enter ID: 1
Error: ПОМИЛКА:  update або delete в таблиці "Brand" порушує обмеження зовнішнього ключа "Brand_FK" таблиці "Product"
DETAIL:  На ключ (Brand_id)=(1) все ще є посилання в таблиці "Product".
```

Завдання 2

Генерація 5 рандомних значень для таблиці Category:

```
Menu:
1. Category
2. SubCategory
3. Brand
4. Product
5. Generate
6. Search
7. Reset all
8. Quit
Enter your choice: 5
Select the amount of data to generate.
Enter number: 5

SubMenu:
1. Category
2. SubCategory
3. Brand
4. Product
5. SubCategory_Brand
6. Quit
Enter your choice: 1
Randomize data added successfully!
```

```
Sub Menu Category :
1. Add Category
2. View Categories
3. Update Category
4. Delete Category
5. Quit
Enter your choice: 2
Categories:
Category ID: 1, Name: Category1
Category ID: 2, Name: Category2
Category ID: 3, Name: Category3
Category ID: 4, Name: Category4
Category ID: 5, Name: Category5
```

Аналогічна операція для таблиці Subcategory:

```
Menu:
1. Category
2. SubCategory
3. Brand
4. Product
5. Generate
6. Search
7. Reset all
8. Quit
Enter your choice: 5
Select the amount of data to generate.
Enter number: 10

SubMenu:
1. Category
2. SubCategory
3. Brand
4. Product
5. SubCategory_Brand
6. Quit
Enter your choice: 2
Randomize data added successfully!
```

```
Sub Menu SubCategory :
1. Add SubCategory
2. View SubCategories
3. Update SubCategory
4. Delete SubCategory
5. Quit
Enter your choice: 2
SubCategories:
SubCategory ID: 1, Name: SubCategory1, Category ID: 4
SubCategory ID: 2, Name: SubCategory2, Category ID: 2
SubCategory ID: 3, Name: SubCategory3, Category ID: 3
SubCategory ID: 4, Name: SubCategory4, Category ID: 3
SubCategory ID: 5, Name: SubCategory5, Category ID: 1
SubCategory ID: 6, Name: SubCategory6, Category ID: 2
SubCategory ID: 7, Name: SubCategory7, Category ID: 1
SubCategory ID: 8, Name: SubCategory8, Category ID: 1
SubCategory ID: 9, Name: SubCategory9, Category ID: 3
SubCategory ID: 10, Name: SubCategory10, Category ID: 2
```

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

```
Brand ID: 1, Name: Brand1
Brand ID: 2, Name: Brand2
Brand ID: 3, Name: Brand3
Brand ID: 4, Name: Brand4
Brand ID: 5, Name: Brand5
Brand ID: 6, Name: Brand6
Brand ID: 7, Name: Brand7
Brand ID: 8, Name: Brand8
Brand ID: 9, Name: Brand9
Brand ID: 10, Name: Brand10
```

```
Product ID: 2, Name: Product2, Color: Red, Width: 70, Height: 86, Deepth: 78, Weight: 84, Energy consumption: 48, Brand_id: 1
Product ID: 3, Name: Product3, Color: Green, Width: 22, Height: 69, Deepth: 37, Weight: 37, Energy consumption: 47, Brand_id: 10
Product ID: 4, Name: Product4, Color: Red, Width: 6, Height: 26, Deepth: 8, Weight: 60, Energy consumption: 74, Brand_id: 2
Product ID: 5, Name: Product5, Color: Blue, Width: 16, Height: 46, Deepth: 38, Weight: 39, Energy consumption: 20, Brand_id: 8
Product ID: 6, Name: Product6, Color: Green, Width: 43, Height: 55, Deepth: 16, Weight: 86, Energy consumption: 69, Brand_id: 6
Product ID: 7, Name: Product7, Color: Purple, Width: 48, Height: 66, Deepth: 99, Weight: 3, Energy consumption: 62, Brand_id: 6
Product ID: 8, Name: Product8, Color: Green, Width: 67, Height: 40, Deepth: 72, Weight: 82, Energy consumption: 2, Brand_id: 9
Product ID: 9, Name: Product9, Color: Red, Width: 19, Height: 96, Deepth: 45, Weight: 17, Energy consumption: 21, Brand_id: 6
Product ID: 10, Name: Product10, Color: Red, Width: 56, Height: 54, Deepth: 2, Weight: 79, Energy consumption: 52, Brand_id: 1
Product ID: 11, Name: Product11, Color: Blue, Width: 33, Height: 96, Deepth: 15, Weight: 26, Energy consumption: 26, Brand_id: 10
Product ID: 12, Name: Product12, Color: Red, Width: 83, Height: 48, Deepth: 14, Weight: 43, Energy consumption: 29, Brand_id: 3
Product ID: 13, Name: Product13, Color: Yellow, Width: 53, Height: 42, Deepth: 41, Weight: 61, Energy consumption: 86, Brand_id: 8
Product ID: 14, Name: Product14, Color: Blue, Width: 5, Height: 20, Deepth: 27, Weight: 24, Energy consumption: 51, Brand_id: 4
Product ID: 15, Name: Product15, Color: Yellow, Width: 66, Height: 36, Deepth: 16, Weight: 27, Energy consumption: 32, Brand_id: 9
```

Query | Query History

```
1  DO $$
2              DECLARE
3                  subcategory_id  INT;
4              BEGIN
5                  FOR subcategory_id  IN 1..20
6                  LOOP
7                      INSERT INTO public."Category" ("Category_id", "Name")
8                      VALUES (
9                            nextval('public.category_id'),
10                           'Category' || currval('public.category_id')
11                     )
12                     ON CONFLICT ("Name") DO NOTHING;
13                 END LOOP;
14             END $$;
```

Data Output | Messages | Notifications

```
DO

Query returned successfully in 33 msec.
```

Query | Query History

```
DO $$
DECLARE
    subcategory_id INT;
BEGIN
    FOR subcategory_id IN 1..40
    LOOP
        INSERT INTO public."Category" ("Category_id", "Name")
        VALUES (
            nextval('public.category_id'),
            'Category' || currval('public.category_id')
        )
        ON CONFLICT ("Name") DO NOTHING;
    END LOOP;
END $$;

```

Data Output | Messages | Notifications

```
DO

Query returned successfully in 34 msec.
```

Query | Query History

```
DO $$
            DECLARE
                brand_id  INT;
            BEGIN
                FOR brand_id  IN 1..10
                LOOP
                    INSERT INTO public."Brand" ("Brand_id", "Name")
                    VALUES (
                          nextval('public.brand_id'),
                          'Brand' || currval('public.brand_id')
                     )
                      ON CONFLICT ("Name") DO NOTHING;
                END LOOP;
            END $$;
```

Data Output | Messages | Notifications

```
DO

Query returned successfully in 33 msec.
```

```
DO $$
        DECLARE
            product_id INT;
        BEGIN
            FOR product_id IN 1..6
            LOOP
                INSERT INTO public."Product" ("Product_id", "Name", "Color", "Width", "Height", "Deepth", "Weight", "Energy consumption", "Brand_i
                VALUES (
                  nextval('public.product_id'),
                  'Product' || currval('public.product_id'),
                    CASE WHEN random() < 0.2 THEN 'Red'
                         WHEN random() < 0.4 THEN 'Blue'
                         WHEN random() < 0.6 THEN 'Green'
                         WHEN random() < 0.8 THEN 'Yellow'
                         ELSE 'Purple' END,
                    (random() * 100 + 1)::integer,
                    (random() * 100 + 1)::integer,
                    (random() * 100 + 1)::integer,
                    (random() * 100 + 1)::integer,
                    (random() * 100 + 1)::integer,
                    (SELECT "Brand_id" FROM public."Brand" ORDER BY random() LIMIT 1)
                );
            END LOOP;
        END $$;
```

Data Output Messages Notifications

```
DO

Query returned successfully in 33 msec.
```

Query Query History

```
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 10;
```

Data Output Messages Notifications

```
INSERT 0 10

Query returned successfully in 35 msec.
```

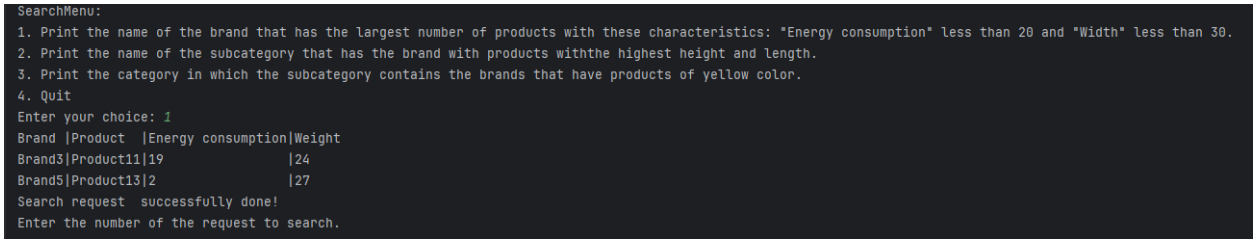
Приклад генерації великої кількості даних:

```
Category ID: 1099988, Name: Category1099988
Category ID: 1099989, Name: Category1099989
Category ID: 1099990, Name: Category1099990
Category ID: 1099991, Name: Category1099991
Category ID: 1099992, Name: Category1099992
Category ID: 1099993, Name: Category1099993
Category ID: 1099994, Name: Category1099994
Category ID: 1099995, Name: Category1099995
Category ID: 1099996, Name: Category1099996
Category ID: 1099997, Name: Category1099997
Category ID: 1099998, Name: Category1099998
Category ID: 1099999, Name: Category1099999
Category ID: 1100000, Name: Category1100000
```

є

Завдання 3

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




```
Enter your choice: 1
Brand |Product  |Energy consumption|Weight
Brand3|Product11|19                |24
Brand5|Product13|2                 |27
Search request  successfully done!
Enter the number of the request to search.
```

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

```
Search request successfully done! Execution time: 11.05 milliseconds
Enter the number of the request to search.
```

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

```
Search request successfully done! Execution time: 12.04 milliseconds
Enter the number of the request to search.
```

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

```
Search request successfully done! Execution time: 10.06 milliseconds
Enter the number of the request to search.
```

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

```
Enter your choice: 2
SubCategoryName|BrandName|ProductName|ProductWidth|ProductHeight
SubCategory1   |Brand5   |Product6   |101         |101
Search request  successfully done!
Enter the number of the request to search.
```

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

```
Enter your choice: 3
Category_Name|SubCategory_Name|Brand_Name|Product_Name
Category5    |SubCategory1    |Brand1    |Product7
Category5    |SubCategory1    |Brand3    |Product16
Category5    |SubCategory1    |Brand5    |Product14
Search request  successfully done!
Enter the number of the request to search.
```

Query Query History

```
1  SELECT b."Name" AS "Brand",
2         p."Name" AS "Product",
3         p."Energy consumption",
4         p."Weight"
5  FROM "public"."Product" p
6  JOIN "public"."Brand" b ON p."Brand_id" = b."Brand_id"
7  WHERE p."Energy consumption" < 20 AND p."Weight" < 30
8  ORDER BY b."Name";
```

Data Output Messages Notifications

	Brand character varying (30)	Product character varying (30)	Energy consumption integer	Weight integer
1	Brand3	Product11	19	24
2	Brand5	Product13	2	27

Query | Query History

```sql
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"
);
```

Data Output | Messages | Notifications

	SubCategoryName character varying (30)	BrandName character varying (30)	ProductName character varying (30)	ProductWidth integer	ProductHeight integer
1	SubCategory1	Brand5	Product6	101	101

Query | Query History

```sql
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';
```

Data Output | Messages | Notifications

	Category_Name character varying (30)	SubCategory_Name character varying (30)	Brand_Name character varying (30)	Product_Name character varying (30)
1	Category5	SubCategory1	Brand1	Product7
2	Category5	SubCategory1	Brand3	Product16
3	Category5	SubCategory1	Brand5	Product14

Завдання 4

```python
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 створює 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.