НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування та спеціалізованих комп'ютерних систем

Розрахункова графічна робота

з дисципліни **Бази даних і засоби управління**

на тему: " Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL "

Виконав: студент групи КВ-23

Паламарчук Максим

Телеграм: https://t.me/Maxim_Pal

Перевірив: Петрашенко А.В.

Київ – 2024

https://github.com/HomerLund/BD_RGR.git

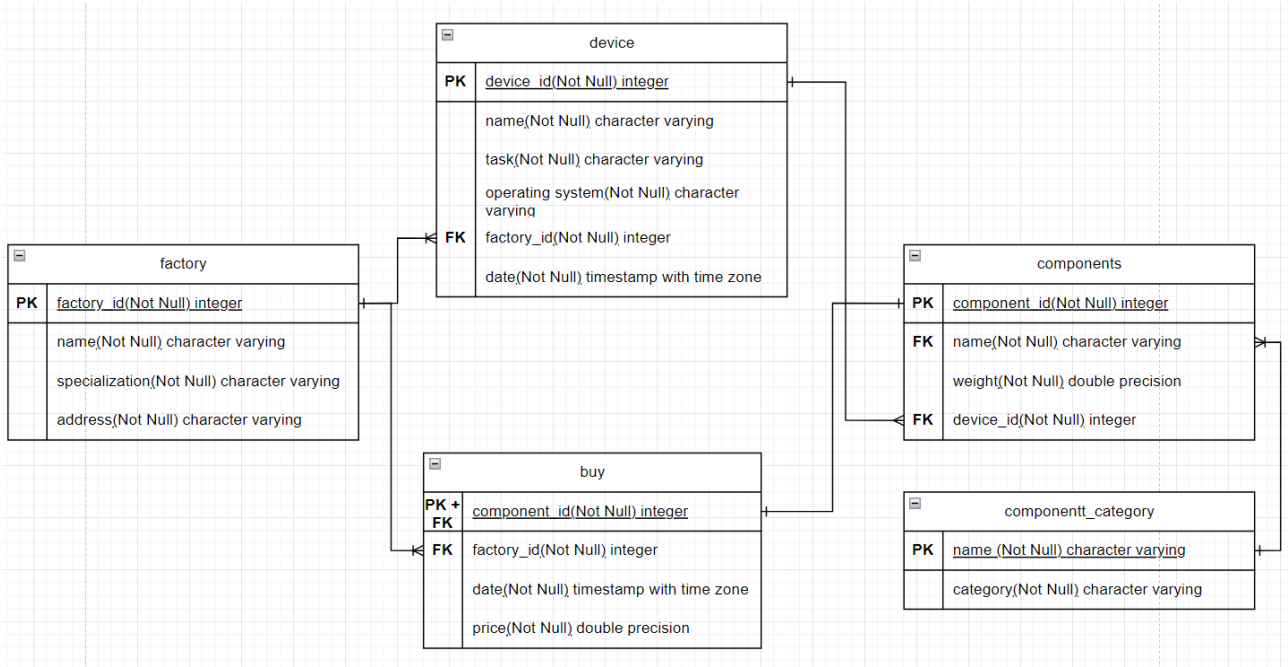
Метою роботи є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

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).

Виконання роботи

Логічна модель (схема) “Система управління та аналізу даних в галузі робототехніки”



Опис сутностей предметної області

1. Фабрика (factory) має атрибути: код фабрики, назва фабрики, спеціалізацію, адресу. Ця сутність описує фабрику та її місцезнаходження.

2. Пристрій (device) має атрибути: код пристрою, назва пристрою, задача, операційна система, код фабрики, дата виготовлення. Ця сутність описує пристрої, які виготовляє фабрика.

3. Комплектуючі (components) має атрибути: код компонента, назва компонента, вага, категорія, код пристрою. Ця сутність описує комплектуючі, які замовляє фабрика для виготовлення пристроїв.

4. Купівля (buy) має атрибути: код компонента, код фабрики, дата купівлі, ціна. Ця сутність описує купівлю комплектуючих заводом.

Опис зв’язків між сутностями предметної області

Сутність factory має зв’язок 1:M по відношенню до сутності device, оскільки один завод може виготовити багато пристроїв, але один пристрій може бути виготовлений тільки однією фабрикою.

Сутність device має зв’язок 1:M по відношенню до сутності components, оскільки один пристрій може бут зібраний із багатьох комплектуючих, але один компонент не може бути в декількох пристроях.

Сутність components має зв’язок 1:1 по відношенню до сутності buy, оскільки один компонент може бути куплений лише один раз.

Сутність factory має зв’язок 1:M по відношенню до сутності buy, оскільки одна фабрика може зробити багато покупок, але одна купівля може відноситися лише до однієї фабрики.

Середовище та компоненти розробки

Для розробки використовувалась мова програмування Python, середовище розробки Visual Studio, а також бібліотека, що надає API для доступу до PostgreSQL – psycopg.

Шаблон проектування

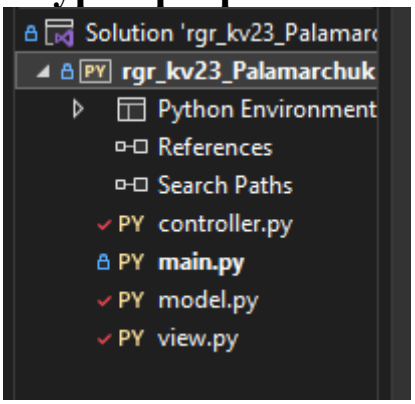
MVC - шаблон проектування, який використаний у програмі.

Model – представляє клас, що описує логіку використовуваних даних. Згідно компоненту моделі, у моїй програмі відповідають всі компоненти які знаходяться у папці Models.

View – в нашому випадку консольний інтерфейс з яким буде взаємодіяти наш користувач. Згідно компоненту представлення, то їй відповідають такі компоненти, згідно яким користувач бачить необхідні дані, що є представленням даних у вигляді консольного інтерфейсу.

Controller – представляє клас, що забезпечує зв'язок між користувачем і системою, поданням і сховищем даних. Він отримує вводяться користувачем дані і обробляє їх. І в залежності від результатів обробки відправляє користувачеві певний висновок, наприклад, у вигляді подання.

Структура програми та її опис



Програма умовно розділена на 4 модулі: файл controller.py, файл model.py та файл view.py та головний файл main.py

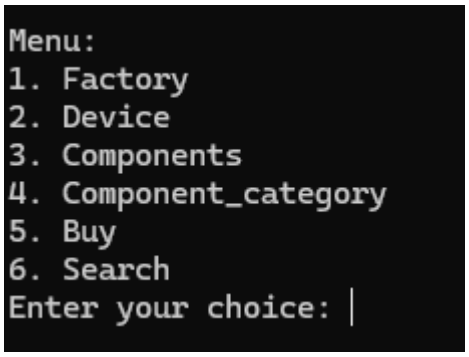
Класи, як видно з їх назв, повністю відповідають використаному патерну MVC.

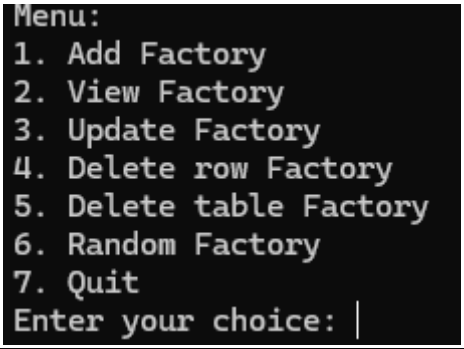
У файлі model описаний клас моделі, що займається регулюванням підключення до бази даних, та виконанням запитів до неї.

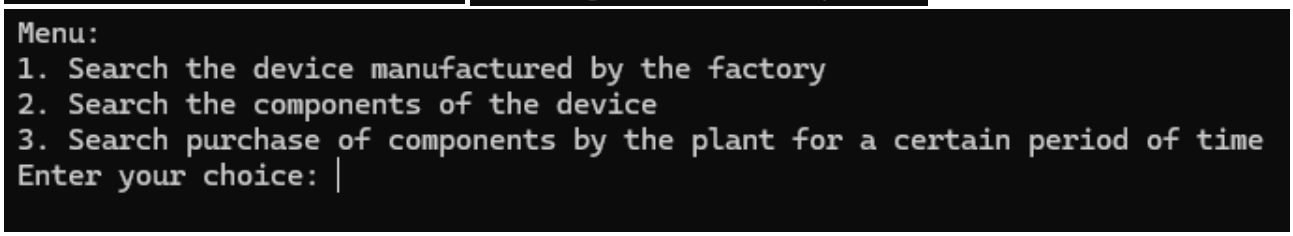
У файлі controller описаний інтерфейс взаємодії з користувачем, запит бажаної дії, виконання пошуку, тощо.

У файлі view описаний клас, що виводить результати виконання тієї чи іншої дії на екран консолі.

Структура меню програми







Меню складається із трьох додаткових меню: меню для вибору таблиці, меню для виконання запитів в таблиці, меню для виконання пошуку в таблицях.

Меню для вибору таблиці:

1. Factory – вибір таблиці Factory

2. Device – вибір таблиці Device

3. Components – вибір таблиці Components

4. Component_category – вибір таблиці Component_category

5. Buy – вибір таблиці Buy

6. Search – Пошук по таблицям

Меню для виконання запитів в таблиці:
1. Add Table – додавання даних до таблиці

2. View Table – перегляд даних таблиці

3. Update Table – зміна даних в таблиці

4. Delete row Table – видалення рядка таблиці

5. Delete all rows Table – видалення всіх рядків таблиці

6. Random Table – генерування рандомізованих даних в таблиці

7. Quit – вихід із програми

Меню для виконання пошуку в таблицях:

1. Search the device manufactured by the factory – знаходження всіх пристроїв, які виготовляє конкретна фабрика.

2. Search the components of the device – знаходження всіх компонентів, з яких складається конкретний пристрій.

3. Search purchase of components by the plant for a certain period of time – знаходження всіх покупок конкретної фабрики за певний період часу.

Фрагменти програм внесення, редагування та вилучення даних у базі даних

Фрагмент програми для введення даних в таблицю:

```
def add_row(self, attributes, attributes_name, table):
    c = self.conn.cursor()
    quary = ''
    s = ''
    for name in attributes_name:
        quary += name + ', '
        s += '%s, '
    quary = quary.rstrip(', ')
    s = s.rstrip(', ')

    query = 'INSERT INTO ' + table.lower() + ' (' + quary + ') VALUES (' + s + ')'
    values = attributes

    c.execute(query, values)

    self.conn.commit()
```

Фрагмент програми для видалення даних з таблиці:

```
def delete_row(self, row_id, PK, table):
    c = self.conn.cursor()
    c.execute('DELETE FROM ' + table.lower() + ' WHERE ' + PK + '=%s', (row_id,))

    self.conn.commit()
```

Фрагмент програми для оновлення даних в таблиці:

```
def update_row(self, row_id, PK, attributes, attributes_name, table):
    c = self.conn.cursor()

    quary = ''
    for name in attributes_name:
        quary += name + '=%s, '
    quary = quary.rstrip(', ')

    query = 'UPDATE ' + table.lower() + ' SET ' + quary + ' WHERE ' + PK + '=%s'
    values = attributes
    values.append(row_id)

    c.execute(query, values)

    self.conn.commit()
```

Фрагмент програми для генерування даних в таблиці:

```
def random_table(self, counts, table):
    c = self.conn.cursor()
    query = 'CALL random_' + table.lower() + '(' + str(counts) + ')'
    c.execute(query)

    self.conn.commit()
```

Фрагмент програми для пошуку даних у таблиці:

```
def get_DeviceOfFactory(self, FK):
    c = self.conn.cursor()

    start_time = time.time()

    query = """ SELECT f.factory_id,  f.name, f.address, d.name, COUNT(*) as count
FROM device d
                JOIN factory f ON d.factory_id = f.factory_id
                WHERE f.factory_id = """ + str(FK) + """
                GROUP BY f.factory_id, d.name
                ORDER BY f.factory_id ASC;"""
    c.execute(query)

    end_time = time.time()
    duration = end_time - start_time

    rows = c.fetchall()

    column_names = [desc[0] for desc in c.description]

    return rows, column_names, duration * 1000

def get_ComponentsOfDevice(self, FK):
    c = self.conn.cursor()

    start_time = time.time()

    query = """ SELECT d.device_id,  d.name, c.name, AVG(c.weight) as avg_weight FROM
device d
                JOIN components c ON d.device_id = c.device_id
                WHERE d.device_id = """ + str(FK) + """
                GROUP BY d.device_id, c.name
                ORDER BY d.device_id ASC;"""
    c.execute(query)

    end_time = time.time()
    duration = end_time - start_time

    rows = c.fetchall()

    column_names = [desc[0] for desc in c.description]

    return rows, column_names, duration * 1000

def get_BuyOfComponents(self, first_date, second_date, FK):
    c = self.conn.cursor()

    start_time = time.time()

    query = """ SELECT f.factory_id,  f.name, c.component_id, c.name, b.date, b.price
FROM buy b
                JOIN factory f ON b.factory_id = f.factory_id
                JOIN components c ON b.component_id = c.component_id
                WHERE b.date BETWEEN '""" + first_date + """' AND '""" + second_date +
"""'
                AND f.factory_id = """ + str(FK) + """
                ORDER BY f.factory_id ASC;"""
    c.execute(query)

    end_time = time.time()
    duration = end_time - start_time

    rows = c.fetchall()

    column_names = [desc[0] for desc in c.description]
```

```
    return rows, column_names, duration * 1000
```

Фрагмент програми для отримання імен стовпчиків таблиці:

```
def get_attributes(self, table):
    c = self.conn.cursor()
    c.execute(f"SELECT * FROM " + table.lower() + " LIMIT 0")
    return [desc[0] for desc in c.description]
```

Скріншоти результатів виконання операції вставки запису в дочірню таблицю

При успішній вставці:

```
Menu:
1. Factory
2. Device
3. Components
4. Component_category
5. Buy
6. Search
Enter your choice: 2

Menu:
1. Add Device
2. View Device
3. Update Device
4. Delete row Device
5. Delete table Device
6. Random Device
7. Quit
Enter your choice: 1
Enter Device name: dev1
Enter Device task: task1
Enter Device operating_system: os1
Enter Device factory_id: 1
Enter Device date: 2024-01-01 00:00:00
Device added successfully!
```

	factory_id [PK] integer	name character varying (30)	specialization character varying (60)	address character varying (70)
1	1	name1	spec1	adres1

	device_id [PK] integer	name character varying (20)	task character varying (200)	operating_system character varying (20)	factory_id integer	date timestamp with time zone
1	1	dev1	task1	os1	1	2024-01-01 00:00:00+02

Коли в таблиці factory немає поля з factory_id = 15:

```
Menu:
1. Factory
2. Device
3. Components
4. Component_category
5. Buy
6. Search
Enter your choice: 2

Menu:
1. Add Device
2. View Device
3. Update Device
4. Delete row Device
5. Delete table Device
6. Random Device
7. Quit
Enter your choice: 1
Enter Device name: name1
Enter Device task: task2
Enter Device operating_system: os2
Enter Device factory_id: 15
Enter Device date: 2024-01-01 00:00:00
Error: insert або update в таблиці "device" порушує обмеження зовнішнього ключа "factory_device"
DETAIL:  Ключ (factory_id)=(15) не присутній в таблиці "factory".
```

	factory_id [PK] integer	name character varying (30)	specialization character varying (60)	address character varying (70)
1	1	name1	spec1	adres1

	device_id [PK] integer	name character varying (20)	task character varying (200)	operating_system character varying (20)	factory_id integer	date timestamp with time zone
1	1	dev1	task1	os1	1	2024-01-01 00:00:00+02

Скріншоти результатів виконання операції видалення даних з таблиці

Результат в батьківській і дочірній таблицях:

	factory_id [PK] integer	name character varying (30)	specialization character varying (60)	address character varying (70)
1	1	name1	spec1	adres1

	device_id [PK] integer	name character varying (20)	task character varying (200)	operating_system character varying (20)	factory_id integer	date timestamp with time zone
1	1	dev1	task1	os1	1	2024-01-01 00:00:00+02

```
Menu:
1. Factory
2. Device
3. Components
4. Component_category
5. Buy
6. Search
Enter your choice: 1

Menu:
1. Add Factory
2. View Factory
3. Update Factory
4. Delete row Factory
5. Delete table Factory
6. Random Factory
7. Quit
Enter your choice: 4
Enter Factory ID: 1
Factory deleted successfully!
```

	factory_id [PK] integer	name character varying (30)	specialization character varying (60)	address character varying (70)

	device_id [PK] integer	name character varying (20)	task character varying (200)	operating_system character varying (20)	factory_id integer	date timestamp with time zone

Скріншоти результатів виконання операції оновлення даних в таблиці

Перед оновленням:

	factory_id [PK] integer	name character varying (30)	specialization character varying (60)	address character varying (70)
1	2	ABB Robotics	industrial robots for assembly and processing	Address 5
2	3	Boston Dynamics	industrial robots for assembly and processing	Address 3
3	4	ABB Robotics	industrial robots for automation of production process...	Address 2
4	7	ABB Robotics	industrial robots for automation of production process...	Address 4
5	9	ABB Robotics	mobile robots with advanced maneuvering capabilities	Address 1

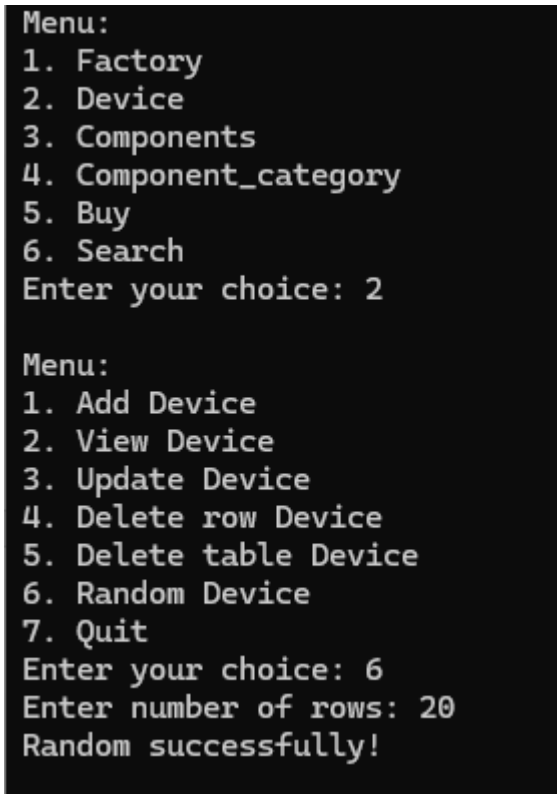
Після:

```
Menu:
1. Factory
2. Device
3. Components
4. Component_category
5. Buy
6. Search
Enter your choice: 1

Menu:
1. Add Factory
2. View Factory
3. Update Factory
4. Delete row Factory
5. Delete table Factory
6. Random Factory
7. Quit
Enter your choice: 3
Enter Factory ID: 2
Enter Factory name: newname
Enter Factory specialization: newspec
Enter Factory address: newadres
Factory updated successfully!
```

	factory_id [PK] integer	name character varying (30)	specialization character varying (60)	address character varying (70)
1	2	newname	newspec	newadres
2	3	Boston Dynamics	industrial robots for assembly and processing	Address 3
3	4	ABB Robotics	industrial robots for automation of production process...	Address 2
4	7	ABB Robotics	industrial robots for automation of production process...	Address 4
5	9	ABB Robotics	mobile robots with advanced maneuvering capabilities	Address 1

Скріншоти результатів виконання операції генерування даних в таблиці



	device_id [PK] integer	name character varying (20)	task character varying (200)	operating_system character varying (20)	factory_id integer	date timestamp with time zone
1	3	robot Spot	universal production processes such as welding, packaging, component handling, and precision material process…	RobotWare	9	2024-04-11 00:00:00+03
2	4	robot Spot	universal production processes such as welding, packaging, component handling, and precision material process…	Spot SDK	2	2024-07-02 00:00:00+03
3	5	robot Spot	heavy lifting operations, welding, material handling, automation of assembly processes	RobotWare	9	2024-03-05 00:00:00+02
4	6	robot Spot	inspection of hard-to-reach or dangerous places, data collection, and assistance in rescue operations	RobotWare	7	2024-06-28 00:00:00+03
5	7	robot IRB 6700	universal production processes such as welding, packaging, component handling, and precision material process…	KUKA System Software	2	2024-10-03 00:00:00+03
6	8	robot KR QUANTEC	inspection of hard-to-reach or dangerous places, data collection, and assistance in rescue operations	RobotWare	7	2024-06-14 00:00:00+03
7	9	robot Spot	universal production processes such as welding, packaging, component handling, and precision material process…	Spot SDK	4	2024-05-30 00:00:00+03
8	10	robot IRB 6700	heavy lifting operations, welding, material handling, automation of assembly processes	Spot SDK	2	2024-08-03 00:00:00+03
9	11	robot IRB 6700	universal production processes such as welding, packaging, component handling, and precision material process…	KUKA System Software	3	2024-01-01 00:00:00+02
10	12	robot IRB 6700	heavy lifting operations, welding, material handling, automation of assembly processes	KUKA System Software	2	2024-03-02 00:00:00+02
11	13	robot Spot	heavy lifting operations, welding, material handling, automation of assembly processes	KUKA System Software	2	2024-07-18 00:00:00+03
12	14	robot Spot	universal production processes such as welding, packaging, component handling, and precision material process…	KUKA System Software	7	2024-08-28 00:00:00+03
13	15	robot Spot	universal production processes such as welding, packaging, component handling, and precision material process…	RobotWare	4	2024-05-25 00:00:00+03

Total rows: 20 of 20 Query complete 00:00:00.053 Ln 2, Col 24

Скріншоти результатів виконання операції пошуку даних у таблиці

```
Menu:
1. Factory
2. Device
3. Components
4. Component_category
5. Buy
6. Search
Enter your choice: 6

Menu:
1. Search the device manufactured by the factory
2. Search the components of the device
3. Search purchase of components by the plant for a certain period of time
Enter your choice: 1
Enter factory_id: 2
Table:
+--------------+---------+-----------+------------------+---------+
|   factory_id | name    | address   | name             |   count |
+==============+=========+===========+==================+=========+
|            2 | newname | newadres  | robot Spot       |       2 |
+--------------+---------+-----------+------------------+---------+
|            2 | newname | newadres  | robot KR QUANTEC |       1 |
+--------------+---------+-----------+------------------+---------+
|            2 | newname | newadres  | robot IRB 6700   |       4 |
+--------------+---------+-----------+------------------+---------+
duration = 1.0013580322265625 milliseconds
```

```
Menu:
1. Factory
2. Device
3. Components
4. Component_category
5. Buy
6. Search
Enter your choice: 6

Menu:
1. Search the device manufactured by the factory
2. Search the components of the device
3. Search purchase of components by the plant for a certain period of time
Enter your choice: 2
Enter device_id: 12
Table:
+-------------+----------------+-----------------+--------------+
|   device_id | name           | name            |   avg_weight |
+=============+================+=================+==============+
|          12 | robot IRB 6700 | Gearboxes 13    |       2.2474 |
+-------------+----------------+-----------------+--------------+
|          12 | robot IRB 6700 | Gearboxes 35    |      43.4723 |
+-------------+----------------+-----------------+--------------+
|          12 | robot IRB 6700 | Servo motors 21 |      57.8077 |
+-------------+----------------+-----------------+--------------+
|          12 | robot IRB 6700 | Servo motors 37 |      68.7459 |
+-------------+----------------+-----------------+--------------+
duration = 1.0027885437011719 milliseconds
```

```
Menu:
1. Factory
2. Device
3. Components
4. Component_category
5. Buy
6. Search
Enter your choice: 6

Menu:
1. Search the device manufactured by the factory
2. Search the components of the device
3. Search purchase of components by the plant for a certain period of time
Enter your choice: 3
Enter factory_id: 4
Enter first date: 2024-02-28 00:00:00+02
Enter second date: 2024-06-02 00:00:00+03
Table:
+-------------+--------------+---------------+----------------+---------------------------+---------+
|   factory_id | name         |   component_id | name           | date                      |   price |
+=============+==============+===============+================+===========================+=========+
|            4 | ABB Robotics |            12 | Servo motors 21 | 2024-02-28 00:00:00+02:00 | 5.84557 |
+-------------+--------------+---------------+----------------+---------------------------+---------+
|            4 | ABB Robotics |            25 | Gearboxes 35    | 2024-06-02 00:00:00+03:00 | 5.38337 |
+-------------+--------------+---------------+----------------+---------------------------+---------+
|            4 | ABB Robotics |            38 | Servo motors 35 | 2024-03-21 00:00:00+02:00 | 3.01744 |
+-------------+--------------+---------------+----------------+---------------------------+---------+
duration = 1.0013580322265625 milliseconds
```

Копії sql запитів для генерації (процедури в базі даних)

random_factory(integer counts)

```
DECLARE
    address_prefix TEXT := 'Address ';
    new_address TEXT;
    random_number INTEGER;
    inserted_count INTEGER := 0;
      existing_count INTEGER;
      last_id INTEGER;
BEGIN
      SELECT COUNT(*) INTO existing_count FROM factory;
      SELECT COALESCE(MAX(factory_id), 0) INTO last_id FROM factory;
    WHILE inserted_count < counts LOOP
        random_number := FLOOR(RANDOM() * (counts + existing_count + 1 - last_id)) +
last_id;
        new_address := address_prefix || random_number;

        BEGIN
            INSERT INTO factory (name, specialization, address)
            VALUES (
                (SELECT name FROM (VALUES ('KUKA Robotics'), ('ABB Robotics'),
('Boston Dynamics')) AS names(name) ORDER BY random() LIMIT 1),
                (SELECT specialization FROM (VALUES ('industrial robots for assembly
and processing'), ('industrial robots for automation of production processes'),
('mobile robots with advanced maneuvering capabilities')) AS
specializations(specialization) ORDER BY random() LIMIT 1),
                new_address
            );

            inserted_count := inserted_count + 1;
        EXCEPTION
            WHEN unique_violation THEN
                CONTINUE;
        END;
    END LOOP;

    RAISE NOTICE 'Total inserted addresses: %', inserted_count;
END;
```

random_device(integer counts)

```
DECLARE
      random_date DATE;
      factory_id_random INTEGER;
      factory_ids INTEGER[];
    random_number INTEGER;
    inserted_count INTEGER := 0;
      existing_count INTEGER;
      last_id INTEGER;
BEGIN
      SELECT ARRAY(SELECT factory_id FROM factory) INTO factory_ids;

      SELECT COUNT(*) INTO existing_count FROM device;
      SELECT COALESCE(MAX(device_id), 0) INTO last_id FROM device;
    WHILE inserted_count < counts LOOP
            random_date := (timestamp '2024-01-01 00:00:00' + random() * (timestamp
'2024-12-30 00:00:00' - timestamp '2024-01-01 00:00:00'))::DATE;
        factory_id_random := factory_ids[FLOOR(random() * array_length(factory_ids, 1)
+ 1)::INTEGER];
            BEGIN
                  INSERT INTO device (name, task, operating_system, factory_id, date)
                  VALUES (
                        (SELECT name FROM (VALUES ('robot KR QUANTEC'), ('robot IRB
6700'), ('robot Spot')) AS names(name) ORDER BY random() LIMIT 1),
                        (SELECT task FROM (VALUES ('universal production processes
such as welding, packaging, component handling, and precision material processing.'),
                        ('heavy lifting operations, welding, material handling,
automation of assembly processes'),
                        ('inspection of hard-to-reach or dangerous places, data
collection, and assistance in rescue operations')) AS tasks(task) ORDER BY random()
LIMIT 1),
                        (SELECT operating_system FROM (VALUES ('KUKA System
Software'), ('RobotWare'), ('Spot SDK')) AS operating_systems(operating_system) ORDER
BY random() LIMIT 1),
                         factory_id_random,
                         random_date
                  );

                  inserted_count := inserted_count + 1;
             EXCEPTION
                  WHEN foreign_key_violation THEN
                        RAISE NOTICE 'Error: there is no such key';
               EXIT;
            WHEN unique_violation THEN
               CONTINUE;
        END;
    END LOOP;

    RAISE NOTICE 'Total inserted addresses: %', inserted_count;
END;
```

random_components(integer counts)

```
DECLARE
    name_random TEXT;
    random_weight DOUBLE PRECISION;
    device_id_random INTEGER;
    random_number INTEGER;
    inserted_count INTEGER := 0;
    existing_count INTEGER;
    last_id INTEGER;
    name_list TEXT[];
    device_id_list INTEGER[];
BEGIN
    SELECT ARRAY(SELECT name FROM component_category) INTO name_list;
    SELECT ARRAY(SELECT device_id FROM device) INTO device_id_list;
```

```
    SELECT COUNT(*) INTO existing_count FROM components;
    SELECT COALESCE(MAX(component_id), 0) INTO last_id FROM components;

    WHILE inserted_count < counts LOOP
        random_weight := random() * 100;
        name_random := name_list[FLOOR(random() * array_length(name_list, 1) +
1)::INTEGER];
        device_id_random := device_id_list[FLOOR(random() *
array_length(device_id_list, 1) + 1)::INTEGER];

        BEGIN
            INSERT INTO components (name, weight, device_id)
            VALUES (name_random, random_weight, device_id_random);

            inserted_count := inserted_count + 1;
        EXCEPTION
            WHEN foreign_key_violation THEN
                RAISE NOTICE 'Error: there is no such key';
                EXIT;
            WHEN unique_violation THEN
                CONTINUE;
        END;
    END LOOP;

    RAISE NOTICE 'Total inserted addresses: %', inserted_count;
END;
```

random_component_category(integer counts)

```
DECLARE
      name_random TEXT;
      random_number INTEGER;
      category_random TEXT;
    inserted_count INTEGER := 0;
      existing_count INTEGER;
      last_id INTEGER;
BEGIN
      SELECT COUNT(*) INTO existing_count FROM component_category;

    WHILE inserted_count < counts LOOP
             random_number := FLOOR(RANDOM() * (counts + existing_count + 1 ));
             BEGIN
                   INSERT INTO component_category (name, category)
                   VALUES (
                         (SELECT name FROM (VALUES ('Gearboxes ' || random_number),
('Servo motors ' || random_number), ('LiDAR sensors ' || random_number)) AS
names(name) ORDER BY random() LIMIT 1),
                         (SELECT category FROM (VALUES ('Mechanical transmissions'),
('Sensors'), ('Drives and motors')) AS categories(category) ORDER BY random() LIMIT 1)

                   );

                   inserted_count := inserted_count + 1;
             EXCEPTION
            WHEN unique_violation THEN
                CONTINUE;
        END;
    END LOOP;

    RAISE NOTICE 'Total inserted addresses: %', inserted_count;
END;
```

random_ buy (integer counts)

```
DECLARE
    random_date DATE;
    random_price DOUBLE PRECISION;
    component_id_random INTEGER;
    factory_id_random INTEGER;
    inserted_count INTEGER := 0;
    existing_count INTEGER;
    last_id INTEGER;
    factory_ids INTEGER[];
    component_ids INTEGER[];
BEGIN
    SELECT ARRAY(SELECT factory_id FROM factory) INTO factory_ids;
    SELECT ARRAY(SELECT component_id FROM components) INTO component_ids;

    SELECT COUNT(*) INTO existing_count FROM buy;
    SELECT COALESCE(MAX(component_id), 0) INTO last_id FROM buy;

    WHILE inserted_count < counts LOOP
        random_date := (timestamp '2024-01-01 00:00:00' + random() * (timestamp '2024-
12-30 00:00:00' - timestamp '2024-01-01 00:00:00'))::DATE;
        random_price := random() * 10;

        factory_id_random := factory_ids[FLOOR(random() * array_length(factory_ids, 1)
+ 1)::INTEGER];
        component_id_random := component_ids[FLOOR(random() *
array_length(component_ids, 1) + 1)::INTEGER];

        BEGIN
            INSERT INTO buy (component_id, factory_id, date, price)
            VALUES (
                component_id_random,
                factory_id_random,
                random_date,
                random_price
            );

            inserted_count := inserted_count + 1;
        EXCEPTION
            WHEN foreign_key_violation THEN
                RAISE NOTICE 'Error: all available components have already been
purchased';
                EXIT;
            WHEN unique_violation THEN
                CONTINUE;
        END;
    END LOOP;

    RAISE NOTICE 'Total inserted rows: %', inserted_count;
END;
```

Копії sql запитів для пошуку (реалізовані в методах класу model)

get_DeviceOfFactory(self, FK):

```
SELECT f.factory_id,  f.name, f.address, d.name, COUNT(*) as count FROM device d
JOIN factory f ON d.factory_id = f.factory_id
WHERE f.factory_id = FK
GROUP BY f.factory_id, d.name
ORDER BY f.factory_id ASC;
```

FK – параметр, який вводить користувач для пошуку всіх пристроїв, які виготовляє конкретна фабрика.

get_ComponentsOfDevice(self, FK):

```
SELECT d.device_id,  d.name, c.name, AVG(c.weight) as avg_weight FROM device d
JOIN components c ON d.device_id = c.device_id
WHERE d.device_id = FK
GROUP BY d.device_id, c.name
ORDER BY d.device_id ASC;
```

FK – параметр, який вводить користувач для пошуку всіх компонентів, з яких складається конкретний пристрій.

get_BuyOfComponents(self, first_date, second_date, FK):

```
SELECT f.factory_id,  f.name, c.component_id, c.name, b.date, b.price FROM buy b
JOIN factory f ON b.factory_id = f.factory_id
JOIN components c ON b.component_id = c.component_id
WHERE b.date BETWEEN '{first_date}' AND '{second_date}'
AND f.factory_id = FK
ORDER BY f.factory_id ASC;
```

FK – параметр, який вводить користувач для пошуку всіх покупок компонентів, які здійснила конкретна фабрика протягом певного періоду (параметри first_date і second_date)

Текст програми:

model.py

```
import psycopg
import time

class Model:
    def __init__(self):
        self.conn = psycopg.connect(
            dbname='lab1',
            user='postgres',
            password='0000',
            host='localhost',
            port=5432
        )
```

#Метод для отримання назв атрибутів таблиці, приймає параметр назву
#таблиці:

```
    def get_attributes(self, table):
        c = self.conn.cursor()
        c.execute(f"SELECT * FROM " + table.lower() + " LIMIT 0")
        return [desc[0] for desc in c.description]
```

#Метод для додавання даних в таблицю, він приймає атрибути, які ввів
#користувач, назви атрибутів таблиці і назву самої таблиці:

```
    def add_row(self, attributes, attributes_name, table):
        c = self.conn.cursor()
        quary = ''
        s = ''
        for name in attributes_name:
            quary += name + ', '
            s += '%s, '
        quary = quary.rstrip(', ')
        s = s.rstrip(', ')

        query = 'INSERT INTO ' + table.lower() + ' (' + quary + ') VALUES (' + s + ')'
        values = attributes

        c.execute(query, values)
        self.conn.commit()
```

#Метод для пошуку всіх пристроїв, які виготовляє конкретна фабрика, він
#приймає параметр FK – factory_id, тобто конкретну фабрику.

```
    def get_DeviceOfFactory(self, FK):
        c = self.conn.cursor()

        start_time = time.time()

        query = """ SELECT f.factory_id,  f.name, f.address, d.name, COUNT(*) as count
FROM device d
                    JOIN factory f ON d.factory_id = f.factory_id
                    WHERE f.factory_id = """ + str(FK) + """
                    GROUP BY f.factory_id, d.name
                    ORDER BY f.factory_id ASC;"""
        c.execute(query)

        end_time = time.time()
        duration = end_time - start_time

        rows = c.fetchall()
```

```
        column_names = [desc[0] for desc in c.description]

        return rows, column_names, duration * 1000
```

#Метод для пошуку всіх компонентів, з яких складається конкретний пристрій, #він приймає параметр FK – device_id:

```
    def get_ComponentsOfDevice(self, FK):
        c = self.conn.cursor()

        start_time = time.time()

        query = """ SELECT d.device_id,  d.name, c.name, AVG(c.weight) as avg_weight
FROM device d
                    JOIN components c ON d.device_id = c.device_id
                    WHERE d.device_id = """ + str(FK) + """
                    GROUP BY d.device_id, c.name
                    ORDER BY d.device_id ASC;"""
        c.execute(query)

        end_time = time.time()
        duration = end_time - start_time

        rows = c.fetchall()

        column_names = [desc[0] for desc in c.description]

        return rows, column_names, duration * 1000
```

#Метод для пошуку всіх покупок, які здійснила конкретна фабрика, він #приймає параметр FK – factory_id, first_date – початкова дата, second_date – #кінцева дата:

```
    def get_BuyOfComponents(self, first_date, second_date, FK):
        c = self.conn.cursor()

        start_time = time.time()

        query = """ SELECT f.factory_id,  f.name, c.component_id, c.name, b.date,
b.price FROM buy b
                    JOIN factory f ON b.factory_id = f.factory_id
                    JOIN components c ON b.component_id = c.component_id
                    WHERE b.date BETWEEN '""" + first_date + """' AND '""" +
second_date + """'
                    AND f.factory_id = """ + str(FK) + """
                    ORDER BY f.factory_id ASC;"""
        c.execute(query)

        end_time = time.time()
        duration = end_time - start_time

        rows = c.fetchall()

        column_names = [desc[0] for desc in c.description]

        return rows, column_names, duration * 1000
```

#Метод для отримання всіх даних таблиці, він приймає параметр назву таблиці:

```
    def get_all_rows(self, table):
        c = self.conn.cursor()
        c.execute('SELECT * FROM ' + table.lower())
        return c.fetchall()
```

#Метод для оновлення даних таблиці, він приймає параметри: id рядка, який #потрібно змінити, назву первинного ключа, змінені користувачем атрибути, #назви колонок таблиці й назву самої таблиці:

```
    def update_row(self, row_id, PK, attributes, attributes_name, table):
        c = self.conn.cursor()

        quary = ''
        for name in attributes_name:
            quary += name + '=%s, '
        quary = quary.rstrip(', ')

        query = 'UPDATE ' + table.lower() + ' SET ' + quary + ' WHERE ' + PK + '=%s'
        values = attributes
        values.append(row_id)

        c.execute(query, values)
        self.conn.commit()
```

#Метод для видалення даних таблиці, він приймає параметри: id рядка, який #потрібно видалити, назву первинного ключа і назву таблиці:

```
    def delete_row(self, row_id, PK, table):
        c = self.conn.cursor()
        c.execute('DELETE FROM ' + table.lower() + ' WHERE ' + PK + '=%s', (row_id,))
        self.conn.commit()
```

#Метод для скидання автоінкременту до 1 в первинного ключа він приймає #параметр назву таблиці:

```
    def reset_identity(self, table):
        c = self.conn.cursor()
        c.execute("""
            SELECT column_name
            FROM information_schema.columns
            WHERE table_name = %s
            AND is_identity = 'YES'
            ORDER BY ordinal_position
            LIMIT 1;
        """, (table.lower(),))
        result = c.fetchone()

        if result:
            identity_column = result[0]
            c.execute(f"ALTER TABLE {table} ALTER COLUMN {identity_column} RESTART
WITH 1;")
```

#Метод для видалення всіх даних таблиці, він приймає параметр назву таблиці:

```
    def delete_table(self, table):
        c = self.conn.cursor()
        query = 'TRUNCATE TABLE ' + table.lower() + ' CASCADE'
        c.execute(query)
        self.reset_identity(table)
        self.conn.commit()
```

#Метод для генерування рандомізованих даних таблиці, він приймає #параметри: кількість згенерованих рядків і назву таблиці:

```
    def random_table(self, counts, table):
        c = self.conn.cursor()
        query = 'CALL random_' + table.lower() + '(' + str(counts) + ')'
        c.execute(query)
        self.conn.commit()
```

#Метод для скасування всіх змін даних з поточної транзакції при виникненні #помилки:

```
def query_rollback(self):
        self.conn.rollback()
```