

Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Лабораторна робота № 2

з дисципліни

«Бази даних та засоби управління»

**Тема: «Створення додатку бази даних, орієнтованого
на взаємодію з СУБД PostgreSQL»**

Виконала студентка групи:
КВ-03 Віннікова У. В.

Перевірив: Петрашенко А. В.

Оцінка:

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).

URL репозиторію з вихідним кодом: <https://github.com/UlianaVinnikova/database.git>

Модель «сутність-зв'язок» предметної галузі «Магазин» («Shop»):

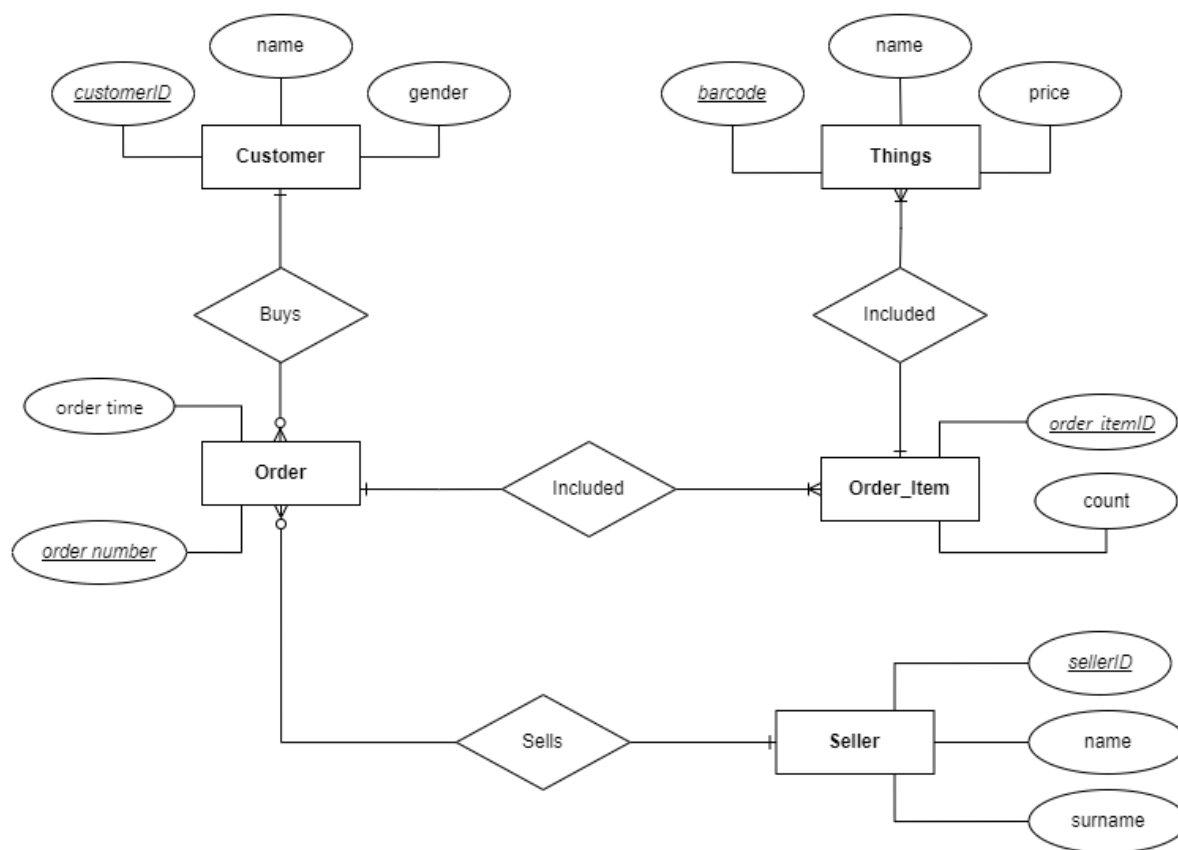


Рисунок 1. ER-діаграма побудована за нотацією «Crow`s foot»

Сутності з описом призначення:

Предметна галузь «Shop» включає в себе 5 сутностей, кожна сутність містить декілька атрибутів:

1. Customer (customerID, name, gender).
2. Seller (sellerID, name, surname).
3. Things (barcode, name, price).
4. Order (order number, order time).
5. Order_Item (order_itemID, count).

Сутність Customer описує покупців, які завітали до даного магазину. Кожен покупець містить інформацію про свій ID, ім'я та стать.

Сутність Seller описує робітників, а точніше продавців магазину. Робітник має унікальний ID, ім'я та прізвище.

Сутність Things відповідає за речі, які продаються в магазині та входять в замовлення. У кожній речі є штрих-код, назва товару та ціна за товар.

Сутність Order це замовлення, яке може купити покупець та продати продавець. Замовлення має власний номер та час замовлення.

Сутність Order_Item є залежною сутністю від Order. Order_Item має ID та відповідає за кількість товару, який входить в замовлення.

Модель «сутність-зв'язок» у схемі бази даних PostgreSQL:

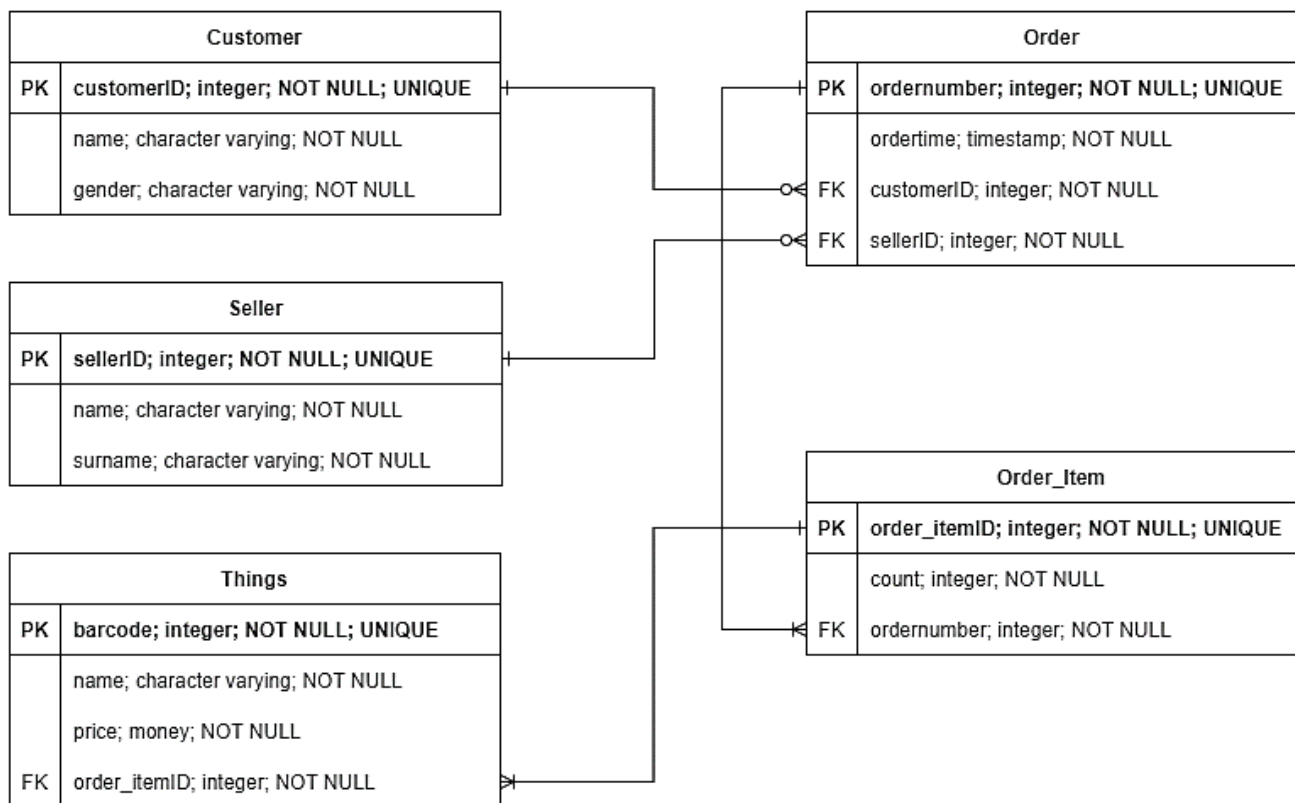


Рисунок 2. Схема бази даних у графічному вигляді

Середовище для відлагодження SQL-запитів до бази даних – PgAdmin4.

Мова програмування – Python 3.10

Середовище розробки програмного забезпечення – PyCharm Community Edition.

Бібліотека взаємодії з PostgreSQL - Psycopg2

MENU

1. Show one table
2. Show all table
3. Insert data
4. Delete data
5. Update data
6. Select data
7. Randomize data
8. Exit

Рисунок 3. Головне меню

Головне меню для користувача складається з восьми пунктів.

Перший пункт (1. Show one table) пропонує виведення однієї таблиці за вибором:

Choose an option: 1

- 1: customer
- 2: seller
- 3: things
- 4: order
- 5: order_item

Choose the table number:

Перед виведенням даних, користувач обирає, яку саме таблицю потрібно вивести.

Після цього на екрані виводяться всі рядки і стовпчики з обраної таблиці БД.

Другий пункт (2. Show all table) пропонує виведення всіх таблиць. Послідовно виводяться усі таблиці БД, після чого користувач знову повертається до головного меню і може обрати нову опцію для взаємодії з таблицями бази даних.

Третій пункт (3. Insert data) пропонує внесення даних:

Choose an option: 3

1: customer

2: seller

3: things

4: order

5: order_item

Choose the table number:

Спочатку потрібно обрати, для якої таблиці буде відбуватися внесення. Тому користувач вводить номер, що відповідає певній таблиці. Після цього користувач вводить дані для кожного атрибуту.

Четвертий пункт (4. Delete data) пропонує видалення даних:

Choose an option: 4

1: customer

2: seller

3: things

4: order

5: order_item

Choose the table number:

Спочатку потрібно обрати, для якої таблиці буде відбуватися видалення. Тому користувач вводить номер, що відповідає певній таблиці. Після цього користувач вводить ідентифікатор рядка, який потрібно видалити. Потім відбувається видалення даних відповідного рядка.

П'ятий пункт (5. Update data) пропонує редагування даних:

Choose an option: 5

1: customer

2: seller

3: things

4: order

5: order_item

Choose the table number:

Спочатку потрібно обрати, для якої таблиці буде відбуватися редагування. Тому користувач вводить номер, що відповідає певній таблиці. Після цього користувач обирає, які саме дані редагувати, вводить нові дані, які записуються в таблицю.

Шостий пункт (6. Select data) пропонує пошук за атрибутами з декількох таблиць:

```
Choose an option: 6
```

1. Show name of things and price of things through the -count- in the order
2. Show ordernumber of order and ordertime of order through the -customerID-
3. Show ordernumber of order and ordertime of order through the -sellerID-

```
Choose the number:
```

Користувач обирає, який запит він хоче виконувати, після чого вводить дані для пошуку. Після введення атрибуту користувач може продовжити пошук або повернутися до головного меню і вивести таблицю за допомогою опції «0» або вивести всі таблиці за допомогою опції «1».

Сьомий пункт (7. Randomize data) пропонує внесення випадкових даних:

```
Choose an option: 7
```

```
1. Customer
```

```
2. Seller
```

```
Choose the table number: 2
```

```
How many rows do you want to insert:
```

Користувач обирає, в яку саме таблицю потрібно внести дані та скільки рядків даних він хоче додати. Після цього відбувається внесення даних у відповідну таблицю.

Восьмий пункт (8. Exit) пропонує вихід з програми. Закривається з'єднання з таблицею бази даних і програма завершується.

```
Choose an option: 8
```

```
PostgreSQL connection is closed!
```

Завдання 1

Перегляд однієї таблиці

PostgreSQL connection is start!

MENU

1. Show one table
2. Show all table
3. Insert data
4. Delete data
5. Update data
6. Select data
7. Randomize data
8. Exit

Choose an option: 1

- 1: customer
- 2: seller
- 3: things
- 4: order
- 5: order_item

Choose the table number: 2

Table seller

SQL query: SELECT * FROM public."seller"

```
-----  
sellerID = 3  
name = Mykola  
surname = Kuzmenko  
-----  
-----  
sellerID = 2  
name = Iryna  
surname = Tkachenko  
-----  
-----  
sellerID = 1  
name = Olga  
surname = Pavlenko  
-----
```

Data output Messages Notifications			
	sellerID [PK] integer	name character varying (20)	surname character varying (20)
1	3	Mykola	Kuzmenko
2	2	Iryna	Tkachenko
3	1	Olga	Pavlenko

Choose the table number: 4

Table order

SQL query: SELECT * FROM public."order"

ordernumber = 534

ordertime = 2022-06-29 17:30:47

customerID = 1

sellerID = 2

ordernumber = 61

ordertime = 2022-03-10 09:21:18

customerID = 3

sellerID = 1

ordernumber = 729

ordertime = 2022-07-18 14:56:02

customerID = 2

sellerID = 3

Data output Messages Notifications				
	ordernumber [PK] integer	ordertime timestamp without time zone	customerID integer	sellerID integer
1	534	2022-06-29 17:30:47	1	2
2	61	2022-03-10 09:21:18	3	1
3	729	2022-07-18 14:56:02	2	3

Перегляд всіх таблиць

Choose an option: 2

Table customer

SQL query: SELECT * FROM public."customer"

customerID = 2

name = Pavlo

gender = male

customerID = 1

name = Maria

gender = female

customerID = 3

name = Natalia

gender = female

Table seller

SQL query: SELECT * FROM public."seller"

sellerID = 3

name = Mykola

surname = Kuzmenko

sellerID = 2

name = Iryna

surname = Tkachenko

sellerID = 1

name = Olga

surname = Pavlenko

Table things

SQL query: SELECT * FROM public."things"

```
-----  
barcode = 789  
name = boots  
price = 1 200,00 ?  
order_itemID = 20  
-----  
-----
```

```
barcode = 456  
name = jacket  
price = 850,50 ?  
order_itemID = 10  
-----  
-----
```

```
barcode = 123  
name = sweater  
price = 399,99 ?  
order_itemID = 30  
-----
```

Table order

SQL query: SELECT * FROM public."order"

```
-----  
ordernumber = 534  
ordertime = 2022-06-29 17:30:47  
customerID = 1  
sellerID = 2  
-----  
-----
```

```
ordernumber = 61  
ordertime = 2022-03-10 09:21:18  
customerID = 3  
sellerID = 1  
-----  
-----
```

```
ordernumber = 729  
ordertime = 2022-07-18 14:56:02  
customerID = 2  
sellerID = 3  
-----
```

Table order_item

SQL query: SELECT * FROM public."order_item"

order_otemID = 20

count = 1

ordernumder = 729

order_otemID = 10

count = 2

ordernumder = 534

order_otemID = 30

count = 3

ordernumder = 61

Внесення даних

Таблиця “Customer” до:

Data output Messages Notifications			
	customerID [PK] integer	name character varying (20)	gender character varying (10)
1	1	Maria	female
2	2	Pavlo	male
3	3	Natalia	female

Choose the table number: *1*

Customer ID = *5*

Customer name = *Darina*

Customer gender = *female*

Таблиця “Customer” після:

Data output Messages Notifications			
	customerID [PK] integer	name character varying (20)	gender character varying (10)
1	1	Maria	female
2	2	Pavlo	male
3	3	Natalia	female
4	5	Darina	female

Таблиця “Seller” до:

Data output Messages Notifications			
	sellerID [PK] integer	name character varying (20)	surname character varying (20)
1	1	Olga	Pavlenko
2	2	Iryna	Tkachenko
3	3	Mykola	Kuzmenko

Choose the table number: 2

Seller ID = 4

Seller name = Max

Seller surname = Gambur

```
SQL query: DO $$ BEGIN IF NOT EXISTS (SELECT "sellerID" FROM "seller" WHERE "sellerID" = '4')
            THEN INSERT INTO "seller"("sellerID", "name", "surname") VALUES ('4','Max','Gambur'); RAISE NOTICE 'added'; ELSE RAISE NOTICE 'already exists'; END IF; END $;
```

SQL query: DO \$\$ BEGIN IF NOT EXISTS (SELECT "sellerID" FROM "seller" WHERE "sellerID" = '4') THEN INSERT INTO "seller"("sellerID", "name", "surname") VALUES ('4','Max','Gambur'); RAISE NOTICE 'added'; ELSE RAISE NOTICE 'This sellerID already exists'; END IF; END \$;

Таблиця “Seller” після:

Data output Messages Notifications				
	sellerID [PK] integer	name character varying (20)	surname character varying (20)	
1	1	Olga	Pavlenko	
2	2	Iryna	Tkachenko	
3	3	Mykola	Kuzmenko	
4	4	Max	Gambur	

Приклад помилки, якщо ми хочемо додати дані, але такий ID вже існує:

Choose the table number: **1**

Customer ID = **2**

Customer name = **Petro**

Customer gender = **male**

['ПОВІДОМЛЕННЯ: This customerID already exists\n']

Do you want to continue working with the 'insert'?

Enter 1 if yes or 2 if no:

Лістинг insert:

```
@staticmethod
def insert_table1(customer_ID, customer_name, customer_gender):
    connection = controller.connection()
    cursor = connection.cursor()
    go_on= True
    while go_on:
        notice = "This customerID already exists"
        insert = """DO $$ BEGIN IF NOT EXISTS (SELECT "customerID" FROM
"customer" WHERE "customerID" = {})
        THEN INSERT INTO "customer"("customerID", "name", "gender")
VALUES ({},{},{}) ; RAISE NOTICE {}; ELSE RAISE NOTICE {}; END IF; END $$;
        """.format(customer_ID, customer_ID, customer_name,
customer_gender, "added", notice)
        go on = False
        #print("SQL query: ", insert)
        cursor.execute(insert)
        connection.commit()
        print(connection.notices)
        cursor.close()
        controller.disconnect(connection)

    @staticmethod
```

```

def insert_table2(seller_ID, seller_name, seller_surname):
    connection = controller.connection()
    cursor = connection.cursor()
    go_on = True
    while go_on:
        notice = "'This sellerID already exists'"
        insert = """DO $$ BEGIN IF NOT EXISTS (SELECT "sellerID" FROM "seller"
WHERE "sellerID" = {})
THEN INSERT INTO "seller"("sellerID", "name", "surname")
VALUES ({} ,{} ,{}); RAISE NOTICE {}; ELSE RAISE NOTICE {}; END IF; END $$;
""".format(seller_ID, seller_ID, seller_name, seller_surname,
'added', notice)
        go_on = False
    #print("SQL query: ", insert)
    cursor.execute(insert)
    connection.commit()
    print(connection.notices)
    cursor.close()
    controller.disconnect(connection)

    @staticmethod
    def insert_table3(things_barcode, things_name, things_price,
things_order_itemID):
        connection = controller.connection()
        cursor = connection.cursor()
        go_on = True
        while go_on:
            notice = "'This barcode already exists'"
            insert = """DO $$ BEGIN IF NOT EXISTS (SELECT "order_itemID" FROM
"order_item" WHERE "order_itemID" = {})
AND NOT EXISTS (SELECT "barcode" FROM "things" WHERE "barcode"
= {}) THEN
INSERT INTO "things"("barcode", "name", "price",
"order_itemID") VALUES ({} ,{} ,{} ,{}); RAISE NOTICE {}; ELSE RAISE NOTICE {}; END
IF; END $$;
""".format(things_order_itemID, things_barcode,
things_barcode, things_name, things_price, things_order_itemID, "added", notice)
            go_on = False
        print("SQL query: ", insert)
        cursor.execute(insert)
        connection.commit()
        print(connection.notices)
        cursor.close()
        controller.disconnect(connection)

    @staticmethod
    def insert_table4(order_ordernumber, order_ordertime, order_customerID,
order_sellerID):
        connection = controller.connection()
        cursor = connection.cursor()
        go_on = True
        while go_on:
            notice = "'This ordernumber already exists'"
            insert = """DO $$ BEGIN IF NOT EXISTS (SELECT "cusromerID" FROM
"customer" WHERE "customerID" = {})
AND NOT EXISTS (SELECT "sellerID" FROM "seller" WHERE
"sellerID" = {}) AND NOT EXISTS (select "ordernumber" from "order" where
"ordernumber" = {}) THEN
INSERT INTO "order"("ordernumber", "ordertime", "customerID",
"sellerID") VALUES ({} ,{} ,{} ,{}); RAISE NOTICE {}; ELSE RAISE NOTICE {}; END IF;

```

```

        END $$;""".format(order_customerID, order_sellerID,
order_ordernumber, order_ordernumber, order_ordertime, order_customerID,
order_sellerID, "'added'", notice)
        go_on = False
        print("SQL query: ", insert)
        cursor.execute(insert)
        connection.commit()
        print(connection.notices)
        cursor.close()
        controller.disconnect(connection)

    @staticmethod
    def insert_table5(orderItem_order_itemID, orderItem_count,
orderItem_ordernumber):
        connection = controller.connection()
        cursor = connection.cursor()
        go_on = True
        while go_on:
            notice = "'This order_itemID already exists'"
            insert = """DO $$ BEGIN IF NOT EXISTS (SELECT "ordernumber" FROM
"order" where "ordernumber" = {})
                        AND NOT EXISTS (SELECT "order_itemID" FROM "order_item" WHERE
"order_itemID" = {}) THEN
                        INSERT INTO "order_item"("order_itemID", "count",
"ordernumber") VALUES ({},{},{},{}) ; 'RAISE NOTICE {}; ELSE RAISE NOTICE {}; END IF;
                        END $$;""".format(orderItem_ordernumber,
orderItem_order_itemID, orderItem_order_itemID, orderItem_count,
orderItem_ordernumber, "'added'", notice)
            go_on = False
            print("SQL query: ", insert)
            cursor.execute(insert)
            connection.commit()
            print(connection.notices)
            cursor.close()
            controller.disconnect(connection)

```

Видалення даних

Таблиця “Things” до:

Data output Messages Notifications				
	barcode [PK] integer	name character varying (20)	price money	order_itemID integer
1	123	sweater	399,99 ?	30
2	456	jacket	850,50 ?	10
3	789	boots	1 200,00 ?	20

Choose the table number: 3

Attribute to delete barcode = 123

SQL query: DELETE FROM "things" WHERE "barcode" = '123';

Таблиця "Things" після:

Data output Messages Notifications				
	barcode [PK] integer	name character varying (20)	price money	order_itemID integer
1	456	jacket	850,50 ?	10
2	789	boots	1 200,00 ?	20

Приклад помилки при видаленні рядка з таким ID, яке є зовнішнім ключем в іншій таблиці:

Choose the table number: 1

Attribute to delete customerID = 1

SQL query: DELETE FROM "customer" WHERE "customerID" = '1';DELETE FROM "order" WHERE "customerID" = '1';

PostgreSQL ERROR! ПОМИЛКА: update або delete в таблиці "customer" порушує обмеження зовнішнього ключа "fk_customerID" таблиці "order"
DETAIL: На ключ (customerID)=(1) все ще є посилання в таблиці "order".

Do you want to continue working with the 'delete'?

Enter 1 if yes or 2 if no:

Лістинг delete:

```
@staticmethod
def delete_table1(customer_ID):
    connection = controller.connection()
    cursor = connection.cursor()
    go_on = True
    while go_on:
        delete = f'DELETE FROM "customer" WHERE "customerID" = {customer_ID};'
        \
            f'DELETE FROM "order" WHERE "customerID" = {customer_ID};'
        go_on = False
    print("SQL query: ", delete)
    cursor.execute(delete)
    connection.commit()
    print(connection.notices)
    cursor.close()
    controller.disconnect(connection)

@staticmethod
```

```

def delete_table2(seller_ID):
    connection = controller.connection()
    cursor = connection.cursor()
    go_on = True
    while go_on:
        delete = f'DELETE FROM "seller" WHERE "sellerID" = {seller_ID};' \
                f'DELETE FROM "order" WHERE "sellerID" = {seller_ID};'
        go_on = False
    print("SQL query: ", delete)
    cursor.execute(delete)
    connection.commit()
    print(connection.notices)
    cursor.close()
    controller.disconnect(connection)

@staticmethod
def delete_table3(things_barcode):
    connection = controller.connection()
    cursor = connection.cursor()
    go_on = True
    while go_on:
        delete = f'DELETE FROM "things" WHERE "barcode" = {things_barcode};'
        go_on = False
    print("SQL query: ", delete)
    cursor.execute(delete)
    connection.commit()
    print(connection.notices)
    cursor.close()
    controller.disconnect(connection)

@staticmethod
def delete_table4(order_ordernumber):
    connection = controller.connection()
    cursor = connection.cursor()
    go_on = True
    while go_on:
        delete = f'DELETE FROM "order_item" WHERE "ordernumber" =
{order_ordernumber};' \
                f'DELETE FROM "order" WHERE "ordernumber" =
{order_ordernumber};'
        go_on = False
    print("SQL query: ", delete)
    cursor.execute(delete)
    connection.commit()
    print(connection.notices)
    cursor.close()
    controller.disconnect(connection)

@staticmethod
def delete_table5(orderItem_order_itemID):
    connection = controller.connection()
    cursor = connection.cursor()
    go_on = True
    while go_on:
        delete = f'DELETE FROM "things" WHERE "order_itemID" =
{orderItem_order_itemID};' \
                f'DELETE FROM "order_item" WHERE "order_itemID" =
{orderItem_order_itemID};'
        go_on = False
    print("SQL query: ", delete)
    cursor.execute(delete)

```

```

connection.commit()
print(connection.notices)
cursor.close()
controller.disconnect(connection)

```

Оновлення даних

Таблиця “Things” до:

Data output

Messages

Notifications

≡+

📄

▼

📋

🗑

🗄

⬇

📈

	barcode [PK] integer	name character varying (20)	price money	order_itemID integer
1	456	jacket	850,50 ?	10
2	789	boots	1 200,00 ?	20

Attribute to update (where) barcode = 789

1: name

2: price

Choose the number of attribute: 2

New value of attribute = 1011,25

Таблиця “Things” після:

Data output

Messages

Notifications

≡+

📄

▼

📋

🗑









🗄

⬇

📈

	barcode [PK] integer	name character varying (20)	price money	order_itemID integer
1	456	jacket	850,50 ?	10
2	789	boots	1 011,25 ?	20

Таблиця “Customer” до:

Data output Messages Notifications			
       			
	customerID [PK] integer	name character varying (20)	gender character varying (10)
1	1	Maria	female
2	2	Pavlo	male
3	3	Natalia	female
4	5	Darina	female

Choose the table number: **1**

Attribute to update (where) customerID = **3**

1: name









2: gender

Choose the number of attribute: **1**

New value of attribute = **Lyudmila**

SQL query: DO \$\$ BEGIN IF EXISTS (SELECT "customerID" FROM "customer" WHERE "customerID" = '3') THEN UPDATE "customer" SET "name" = 'Lyudmila' WHERE "customerID" = '3'; RAISE NOTICE 'updated'; ELSE RAISE NOTICE 'There is nothing to update'; END IF; END \$\$;

Таблиця “Customer” після:

Data output Messages Notifications			
       			
	customerID [PK] integer	name character varying (20)	gender character varying (10)
1	1	Maria	female
2	2	Pavlo	male
3	3	Lyudmila	female
4	5	Darina	female

Таблиця "Order" до:

Data output Messages Notifications

	ordernumber [PK] integer	ordertime timestamp without time zone	customerID integer	sellerID integer
1	61	2022-03-10 09:21:18	3	1
2	534	2022-06-29 17:30:47	1	2
3	729	2022-07-18 14:56:02	2	3

Choose the table number: 4

Attribute to update (where) ordernumber = 534

1: ordertime

Choose the number of attribute: 1

New value of attribute = 2023-01-01 17:48:02

SQL query: DO \$\$ BEGIN IF EXISTS (SELECT "ordernumber" FROM "order"
WHERE "ordernumber" = '534') THEN UPDATE "order" SET "ordertime" = '2023-01-01 17:48:02' WHERE "ordernumber" = '534'; RAISE NOTICE 'updated'; ELSE RAISE NOTICE 'There is nothing to update'; END IF; END \$\$;

Таблиця "Order" після:

Data output Messages Notifications

	ordernumber [PK] integer	ordertime timestamp without time zone	customerID integer	sellerID integer
1	61	2022-03-10 09:21:18	3	1
2	534	2023-01-01 17:48:02	1	2
3	729	2022-07-18 14:56:02	2	3

Таблиця “Orde_itemr” до:

Data output Messages Notifications			
	order_itemID [PK] integer	count integer	ordernumber integer
1	10	2	534
2	20	1	729
3	30	3	61

Choose the table number: 5

Attribute to update (where) order_itemID = 20

1: count

Choose the number of attribute: 1

New value of attribute = 18

Таблиця “Order_item” після:

Data output Messages Notifications			
	order_itemID [PK] integer	count integer	ordernumber integer
1	10	2	534
2	20	18	729
3	30	3	61

Лістинг update:

```
@staticmethod
def update_table1(customer_ID, set):
    connection = controller.connection()
    cursor = connection.cursor()
    restart = True
    while restart:
        notice = "There is nothing to update"
```

```

        update = """DO $$ BEGIN IF EXISTS (SELECT "customerID" FROM "customer"
WHERE "customerID" = {}) THEN
        UPDATE "customer" SET {} WHERE "customerID" = {}; RAISE NOTICE
{}; ELSE RAISE NOTICE {}; END IF;
        END $$;""".format(customer_ID, set, customer_ID, "'updated'",
notice)

        restart = False
        pass
        print("SQL query: ", update)
        cursor.execute(update)
        connection.commit()
        print(connection.notices)
        cursor.close()
        controller.disconnect(connection)
        pass

    @staticmethod
    def update_table2(seller_ID, set):
        connection = controller.connection()
        cursor = connection.cursor()
        restart = True
        while restart:
            notice = "'There is nothing to update'"
            update = """DO $$ BEGIN IF EXISTS (SELECT "sellerID" FROM "seller"
WHERE "sellerID" = {}) THEN
            UPDATE "seller" SET {} WHERE "sellerID" = {}; RAISE NOTICE {};
ELSE RAISE NOTICE {}; END IF; END $$;
            """.format(seller_ID, set, seller_ID, "'updated'", notice)
            restart = False
            pass
            print("SQL query: ", update)
            cursor.execute(update)
            connection.commit()
            print(connection.notices)
            cursor.close()
            controller.disconnect(connection)
            pass

    @staticmethod
    def update_table3(things_barcode, set):
        connection = controller.connection()
        cursor = connection.cursor()
        restart = True
        while restart:
            notice = "'There is nothing to update'"
            update = """DO $$ BEGIN IF EXISTS (SELECT "barcode" FROM "things" WHERE
"barcode" = {})
            THEN UPDATE "things" SET {} WHERE "barcode" = {}; RAISE NOTICE
{}; ELSE RAISE NOTICE {}; END IF; END $$;
            """.format(things_barcode, set, things_barcode, "'updated'",
notice)
            restart = False
            pass
            print("SQL query: ", update)
            cursor.execute(update)
            connection.commit()
            print(connection.notices)
            cursor.close()
            controller.disconnect(connection)
            pass

```

```

@staticmethod
def update_table4(order_ordernumber, set):
    connection = controller.connection()
    cursor = connection.cursor()
    restart = True
    while restart:
        notice = "'There is nothing to update'"
        update = """DO $$ BEGIN IF EXISTS (SELECT "ordernumber" FROM "order"
WHERE "ordernumber" = {})
        THEN UPDATE "order" SET {} WHERE "ordernumber" = {}; RAISE
NOTICE {}; ELSE RAISE NOTICE {}; END IF; END $$;
        """.format(order_ordernumber, set, order_ordernumber,
"'updated'", notice)
        restart = False
    pass
    #print("SQL query: ", update)
    cursor.execute(update)
    connection.commit()
    print(connection.notices)
    cursor.close()
    controller.disconnect(connection)
    pass

@staticmethod
def update_table5(orderItem_order_itemID, set):
    connection = controller.connection()
    cursor = connection.cursor()
    restart = True
    while restart:
        notice = "'There is nothing to update'"
        update = """DO $$ BEGIN IF EXISTS (SELECT "order_itemID" FROM
"order_item" WHERE "order_itemID" = {})
        THEN UPDATE "order_item" SET {} WHERE "order_itemID" = {};
RAISE NOTICE {}; ELSE RAISE NOTICE {}; END IF; END $$;
        """.format(orderItem_order_itemID, set,
orderItem_order_itemID, "'updated'", notice)
        restart = False
    pass
    #print("SQL query: ", update)
    cursor.execute(update)
    connection.commit()
    print(connection.notices)
    cursor.close()
    controller.disconnect(connection)

```


Завдання 2

Генерування «рандомізованих» даних

Choose an option: 7

1. Customer

2. Seller

Choose the table number: 1

How many rows do you want to insert: 100

```
INSERT INTO "customer" ("customerID", "name", "gender")
SELECT generate_series as customerID,
chr(trunc(65 + random()*26)::int)||chr(trunc(65 + random()*26)::int)||chr(trunc(65 +
random()*26)::int),
(array['F', 'M'])[floor(random() * 2 + 1)]
FROM generate_series(5, {count}+5)
```

count – вводиться користувачем, скільки рядків хоче додати користувач в таблицю

Data output Messages Notifications			
	customerID [PK] integer	name character varying (20)	gender character varying (10)
1	1	Maria	female
2	2	Pavlo	male
3	3	Lyudmila	female
4	5	PLU	M
5	6	OGI	M
6	7	WWQ	F
7	8	GZA	F
8	9	OFW	F
9	10	OMT	M
10	11	SLY	M
11	12	XZI	M
12	13	WBX	M
13	14	FEB	M
14	15	XFI	F
15	16	KYX	F
16	17	NOZ	F
17	18	NCB	F
18	19	SHB	F
19	20	ITV	M
20	21	YJB	M
21	22	XHO	F
22	23	RXJ	M

	customerID [PK] integer	name character varying (20)	gender character varying (10)
23	24	QBP	F
24	25	QQG	M
25	26	ORU	F
26	27	ZPX	M
27	28	TWY	F
28	29	OJJ	M
29	30	VYB	F
30	31	HYA	M
31	32	TMN	M
32	33	MDH	F
33	34	NPK	F
34	35	TDO	F
35	36	DGH	M
36	37	TWX	F
37	38	JUR	M
38	39	QGI	F
39	40	QLS	M
40	41	IXC	M
41	42	RIE	F
42	43	XHN	F
43	44	RCP	F
44	45	QWO	F

45	46	BBB	F
46	47	QCH	F
47	48	AZY	M
48	49	VKC	F
49	50	ARX	M
50	51	EGV	F
51	52	JYM	F
52	53	XJB	F
53	54	HDX	F
54	55	EAH	M
55	56	VXJ	M
56	57	YUA	M
57	58	IVH	F
58	59	GWJ	F
59	60	XTB	M
60	61	VTY	F
61	62	CWO	M
62	63	UYX	F
63	64	BWD	F
64	65	SXT	M
65	66	AED	F
66	67	AQC	F

67	68	FJN	F
68	69	EOJ	F
69	70	VJN	F
70	71	JIJ	M
71	72	PEE	F
72	73	DBF	F
73	74	VGf	F
74	75	XPQ	M
75	76	ZGS	M
76	77	UIN	F
77	78	JWJ	M
78	79	MNV	M
79	80	GCJ	M
80	81	SSJ	M
81	82	EGC	F
82	83	JFU	M
83	84	EZK	M
84	85	UHT	F
85	86	ILF	F
86	87	UMO	F
87	88	XEO	F
88	89	FQW	M

88	89	FQW	M
89	90	FUT	M
90	91	DXG	M
91	92	DGT	F
92	93	QHP	F
93	94	PJU	M
94	95	YND	F
95	96	AFE	M
96	97	JOG	F
97	98	YBW	F
98	99	EDK	M
99	100	XCP	F
100	101	RVH	F
101	102	PZW	M
102	103	SHC	M
103	104	RZN	M
104	105	KVI	M

Choose an option: 7

1. Customer

2. Seller

Choose the table number: 2

How many rows do you want to insert: 10

```
INSERT INTO "seller" ("sellerID", "name", "surname")
SELECT generate_series as sellerID,
chr(trunc(65 + random()*26)::int)||chr(trunc(65 + random()*26)::int),
chr(trunc(65 + random()*26)::int)||chr(trunc(65 + random()*26)::int)||chr(trunc(65 +
random()*26)::int)
FROM generate_series(5, {count}+5)
```

Data output Messages Notifications			
	sellerID [PK] integer	name character varying (20)	surname character varying (20)
1	1	Olga	Pavlenko
2	2	Iryna	Tkachenko
3	3	Mykola	Kuzmenko
4	5	VL	AQY
5	6	RF	KMA
6	7	NV	QZI
7	8	WT	GRB
8	9	PZ	ESQ
9	10	TS	FLO
10	11	KF	FNU
11	12	CM	MHE
12	13	JI	EBP
13	14	NM	XWE
14	15	JD	OQI

Лістинг random:

```
@staticmethod
def random(table, count):
    connection = controller.connection()
    cursor = connection.cursor()
    go_on = True
    while go_on:
        if table == 1:
            for i in range(count):
                cursor.execute(f"""INSERT INTO "customer" ("customerID",
"name", "gender")
                                SELECT generate_series as customerID,
                                chr(trunc(65 +
random()*26)::int)||chr(trunc(65 + random()*26)::int)||chr(trunc(65 +
random()*26)::int),
                                (array['F', 'M'])[floor(random() * 2 + 1)]
                                FROM generate_series(5, {count}+5)""")
                go_on = False
            elif table == 2:
                for i in range(count):
                    cursor.execute(f"""INSERT INTO "seller" ("sellerID", "name",
"surname")
                                    SELECT generate_series as sellerID,
                                    chr(trunc(65 + random()*26)::int)||chr(trunc(65
+ random()*26)::int),
                                    chr(trunc(65 + random()*26)::int)||chr(trunc(65
+ random()*26)::int)||chr(trunc(65 + random()*26)::int)
                                    FROM generate_series(5, {count}+5)""")
                    go_on = False
        else:
            print("Please, enter valid number: ")
    print("Insertion of randomized data was successful!")
    connection.commit()
    cursor.close()
    controller.disconnect(connection)
```

Задання 3

Пошук даних

Choose an option: 6

1. Show name of things and price of things through the -count- in the order
2. Show ordernumber of order and ordertime of order through the -customerID-
3. Show ordernumber of order and ordertime of order through the -sellerID-

Choose the number: 1

Enter required count: 2

```
SQL query: SELECT "count", "name", "price" from (SELECT o."count", t."name", t."price"
        FROM "things" t LEFT JOIN "order_item" o ON o."order_itemID" = t."order_itemID"
        WHERE o."count" = '2' GROUP BY o."count", t."name", t."price") AS foo
```

Time of request = 6 ms

```
-----
count = 2
name = jacket
price = 850,50 ?
-----
```

Choose the number: 2

Enter required customerID: 3

```
SQL query: SELECT "customerID", "ordernumber", "ordertime" FROM (SELECT c."customerID", o."ordernumber", o."ordertime"
        FROM "order" o LEFT JOIN "customer" c ON c."customerID" = o."customerID"
        WHERE c."customerID" = '3' GROUP BY c."customerID", o."ordernumber", o."ordertime") AS foo
```

Time of request = 1 ms

```
-----
customerID = 3
ordernumber = 61
ordertime = 2022-03-10 09:21:18
-----
```

Choose the number: 3

Enter required sellerID: 1

```
SQL query: SELECT "sellerID", "ordernumber", "ordertime" FROM (SELECT s."sellerID", o."ordernumber", o."ordertime"
        FROM "order" o LEFT JOIN "seller" s ON s."sellerID" = o."sellerID"
        WHERE s."sellerID" = '1' GROUP BY s."sellerID", o."ordernumber", o."ordertime") AS foo
```

Time of request = 3 ms

```
-----
sellerID = 1
ordernumber = 61
ordertime = 2022-03-10 09:21:18
-----
```

Choose the number: 2

Enter required customerID: 1

```
SQL query: SELECT "customerID", "ordernumber", "ordertime" FROM (SELECT c."customerID", o."ordernumber", o."ordertime"
        FROM "order" o LEFT JOIN "customer" c ON c."customerID" = o."customerID"
        WHERE c."customerID" = '1' GROUP BY c."customerID", o."ordernumber", o."ordertime") AS foo
```

Time of request = 7 ms

```
-----
customerID = 1
ordernumber = 534
ordertime = 2023-01-01 17:48:02
-----
```

Лістинг select:

```
@staticmethod
def select_num1(count):
    connection = controller.connection()
    cursor = connection.cursor()
    select = f"""SELECT "count", "name", "price" from (SELECT o."count",
t."name", t."price"
        FROM "things" t LEFT JOIN "order_item" o ON o."order_itemID" =
t."order_itemID"
        WHERE o."count" = '{count}' GROUP BY o."count", t."name",
t."price") AS foo"""
    print("SQL query: ", select)
    beg = int(time.time() * 1000)
    cursor.execute(select)
    end = int(time.time() * 1000) - beg
    records = cursor.fetchall()
    print(f'Time of request = {end} ms')
    cursor.close()
    controller.disconnect(connection)
    return records

@staticmethod
def select_num2(customerID):
    connection = controller.connection()
    cursor = connection.cursor()
    select = f"""SELECT "customerID", "ordernumber", "ordertime" FROM (SELECT
c."customerID", o."ordernumber", o."ordertime"
        FROM "order" o LEFT JOIN "customer" c ON c."customerID" =
o."customerID"
        WHERE c."customerID" = '{customerID}' GROUP BY c."customerID",
o."ordernumber", o."ordertime") AS foo"""
    print("SQL query: ", select)
    beg = int(time.time() * 1000)
    cursor.execute(select)
    end = int(time.time() * 1000) - beg
    records = cursor.fetchall()
    print(f'Time of request = {end} ms')
    cursor.close()
    controller.disconnect(connection)
    return records

@staticmethod
def select_num3(sellerID):
    connection = controller.connection()
    cursor = connection.cursor()
    select = f"""SELECT "sellerID", "ordernumber", "ordertime" FROM (SELECT
s."sellerID", o."ordernumber", o."ordertime"
        FROM "order" o LEFT JOIN "seller" s ON s."sellerID" = o."sellerID"
        WHERE s."sellerID" = '{sellerID}' GROUP BY s."sellerID",
o."ordernumber", o."ordertime") AS foo"""
    print("SQL query: ", select)
    beg = int(time.time() * 1000)
    cursor.execute(select)
    end = int(time.time() * 1000) - beg
    records = cursor.fetchall()
    print(f'Time of request = {end} ms')
    cursor.close()
    controller.disconnect(connection)
    return records
```

Завдання 4

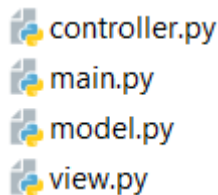
Шаблон MVC

MVC (модель-подання-контролер) – шаблон проектування, який використаний у програмі.

Model – представляє клас, що описує логіку використовуваних даних. Клас реалізований у файлі `model.py`, у ньому відбуваються найважчі процеси (вставка, видалення, оновлення, пошук, рандомізація даних, звернення до бази даних) і після виконаної події відправляє результат до View.

View – це консольний інтерфейс, з яким взаємодіє користувач. Відповідає за введення/виведення даних. У програмі це реалізовано за допомогою файлу `view.py` (клас View та клас Menu).

Controller – забезпечує зв'язок між користувачем і системою, поданням і сховищем даних. Він отримує введені користувачем дані і обробляє їх. У програмі це реалізовано у файлі `controller.py`



main.py – точка входу в програму, запускає початковий інтерфейс.

model.py – виконує операції з базою даних.

view.py – файл, що відповідає за функціонал виведення даних, повідомлень для користувача та реалізовує меню для взаємодії з користувачем, приймає введені дані від користувача і передає їх у контролер.

controller.py – виконує підключення до бази даних, обробляє введені користувачем дані, подає відповідну команду до `model.py`.

Код програми

main.py

```
from view import Menu

print("PostgreSQL connection is start!")

Menu.mainmenu()

print("PostgreSQL connection is closed!")
```

controller.py

```
import psycopg2

def connection():

    return psycopg2.connect(
        user="postgres",
        password="123",
        host="localhost",
        port="5432",
        database="postgres",
    )

def disconnect(connection):
    connection.commit()
    connection.close()

def validate_table(table):
    incorrect = True
    while incorrect:
        if table.isdigit():
            table = int(table)
            if 1 <= table <= 5:
                incorrect = False
            else:
                print("\nERROR!\nIncorrect input,try again!")
        else:
            print("\nERROR!\nIncorrect input,try again!")
    return table
```

view.py

```
import psycopg2
import controller
from model import Model

tables = {
    1: 'customer',
    2: 'seller',
    3: 'things',
    4: 'order',
    5: 'order_item',
```



```
}
```

```
class View:
```

```
    def __init__(self, table, records):  
        self.table = table  
        self.records = records
```

```
@staticmethod
```

```
def list():  
    print('1: customer\n2: seller\n3: things\n4: order\n5: order_item\n')
```

```
@staticmethod
```

```
def attribute_list_for_update(table):  
    if table == 1:
```

```
        print('1: name\n2: gender')
```

```
    elif table == 2:
```

```
        print('1: name\n2: surname')
```

```
    elif table == 3:
```

```
        print('1: name\n2: price')
```

```
    elif table == 4:
```

```
        print('1: ordertime')
```

```
    elif table == 5:
```

```
        print('1: count')
```

```
def show(self):
```

```
    if self.table == 1:
```

```
        for row in self.records:
```

```
            print("-----")
```

```
            print("customerID = ", row[0])
```

```
            print("name = ", row[1])
```

```
            print("gender = ", row[2])
```

```
            print("-----")
```

```
    elif self.table == 2:
```

```
        for row in self.records:
```

```
            print("-----")
```

```
            print("sellerID = ", row[0])
```

```
            print("name = ", row[1])
```

```
            print("surname = ", row[2])
```

```
            print("-----")
```

```
    elif self.table == 3:
```

```
        for row in self.records:
```

```
            print("-----")
```

```
            print("barcode = ", row[0])
```

```
            print("name = ", row[1])
```

```
            print("price = ", row[2])
```

```
            print("order_itemID = ", row[3])
```

```
            print("-----")
```

```
    elif self.table == 4:
```

```
        for row in self.records:
```

```
            print("-----")
```

```
            print("ordernumber = ", row[0])
```

```
            print("ordertime = ", row[1])
```

```
            print("customerID = ", row[2])
```

```
            print("sellerID = ", row[3])
```

```
            print("-----")
```

```
    elif self.table == 5:
```

```

        for row in self.records:
            print("-----")
            print("order_oteID = ", row[0])
            print("count = ", row[1])
            print("ordernumder = ", row[2])
            print("-----")
    else:
        print("\nERROR!\nIncorrect input,try again!")

def show_select(self):
    if self.table == 1:
        for row in self.records:
            print("-----")
            print("count = ", row[0])
            print("name = ", row[1])
            print("price = ", row[2])
            print("-----")
    elif self.table == 2:
        for row in self.records:
            print("-----")
            print("customerID = ", row[0])
            print("ordernumber = ", row[1])
            print("ordertime = ", row[2])
            print("-----")
    elif self.table == 3:
        for row in self.records:
            print("-----")
            print("sellerID = ", row[0])
            print("ordernumber = ", row[1])
            print("ordertime = ", row[2])
            print("-----")

```

```

class Menu:

```

```

    @staticmethod
    def mainmenu():
        exit = False
        while not exit:
            print("\nMENU\n1. Show one table\n2. Show all table\n3. Insert data\n4.
Delete data\n5. Update data\n6. Select data\n7. Randomize data\n8. Exit\n")
            choice = input('Choose an option: ')
            if choice == '1':
                View.list()
                table = input('Choose the table number: ')
                table = controller.validate_table(table)
                records = Model.show_table(table)
                obj = View(table, records)
                obj.show()

            elif choice == '2':
                for i in range(1, 6, 1):
                    records = Model.show_table(i)
                    obj = View(i, records)
                    obj.show()

            elif choice == '3':
                end_insert = False
                while not end_insert:
                    try:
                        View.list()

```

```

table = input('Choose the table number: ')
table = controller.validate_table(table)
if table == 1:
    customer_ID = "" + input("Customer ID = ") + ""
    customer_name = "" + input("Customer name = ") + ""
    customer_gender = "" + input("Customer gender = ") + ""

    Model.insert_table1(customer_ID, customer_name,
customer_gender)

elif table == 2:
    seller_ID = "" + input("Seller ID = ") + ""
    seller_name = "" + input("Seller name = ") + ""
    seller_surname = "" + input("Seller surname = ") + ""
    Model.insert_table2(seller_ID, seller_name,
seller_surname)

elif table == 3:
    things_barcode = "" + input("Things barcode = ") + ""
    things_name = "" + input("Things name = ") + ""
    things_price = "" + input("Things price = ") + ""
    things_order_itemID = "" + input("Things order_itemID
= ") + ""

    Model.insert_table3(things_barcode, things_name,
things_price,things_order_itemID)
elif table == 4:
    order_ordernumber = "" + input("Order ordernumber = ")
+ ""

    order_ordertime = "" + input("Order ordertime = ") +
""

    order_customerID = "" + input("Order customerID = ") +
""

    order_sellerID = "" + input("Order sellerID = ") + ""
    Model.insert_table4(order_ordernumber, order_ordertime,
order_customerID, order_sellerID)
elif table == 5:
    orderItem_order_itemID = "" + input("Order_item
order itemID = ") + ""

    orderItem_count = "" + input("Order_item count = ") +
""

    orderItem_ordernumber = "" + input("Order_item
ordernumber = ") + ""

    Model.insert table5(orderItem_order_itemID,
orderItem_count, orderItem_ordernumber)
else:
    print('ERROR!\nIncorrect input, try again!')
except(Exception, psycopg2.Error) as error:
    print('PostgreSQL ERROR! ',error)
incorrect = True
while incorrect:
    answer = input("Do you want to continue working with the
'insert'? \nEnter 1 if yes or 2 if no: ")
    if answer == '2':
        end_insert = True
        incorrect = False
    elif answer == '1':
        incorrect = False
    pass
else:
    print('ERROR!\nPlease, enter your answer!')

elif choice == '4':
    end_delete = False

```

```

while not end_delete:
    try:
        View.list()
        table = input('Choose the table number: ')
        table = controller.validate_table(table)
        if table == 1:
            customer_ID = "" + input('Attribute to delete
customerID = ') + ""

            Model.delete_table1(customer_ID)
        elif table == 2:
            seller_ID = "" + input('Attribute to delete sellerID =
') + ""

            Model.delete_table2(seller_ID)
        elif table == 3:
            things_barcode = "" + input('Attribute to delete
barcode = ') + ""

            Model.delete_table3(things_barcode)
        elif table == 4:
            order_ordernumber = "" + input('Attribute to delete
ordernumber = ') + ""

            Model.delete_table4(order_ordernumber)
        elif table == 5:
            orderItem_order_itemID = "" + input('Attribute to
delete order_itemID = ') + ""

            Model.delete_table5(orderItem_order_itemID)
        else:
            print('ERROR!\nIncorrect input, try again!')
    except (Exception, psycopg2.Error) as error:
        print('PostgreSQL ERROR! ',error)
    incorrect = True
    while incorrect:
        answer = input("Do you want to continue working with the
'delete'? \nEnter 1 if yes or 2 if no: ")
        if answer == '2':
            end_delete = True
            incorrect = False
        elif answer == '1':
            incorrect = False
        pass
    else:
        print('ERROR!\nPlease, enter your answer!')

elif choice == '5':
    end_update = False
    while not end_update:
        try:
            View.list()
            table = input('Choose the table number: ')
            table = controller.validate_table(table)
            if table == 1:
                customer_ID = "" + input('Attribute to update (where)
customerID = ') + ""

                View.attribute_list_for_update(1)
                in_restart = True
                while in_restart:
                    answer = input('Choose the number of attribute: ')
                    if answer == '1':
                        value = "" + input('New value of attribute =
') + ""

                        set = "name" = {}".format(value)
                        in_restart = False

```

```

        elif answer == '2':
            value = "" + input('New value of attribute = ') + ""

            set = "gender" = {}.format(value)
            in_restart = False
        else:
            print('Incorrect input, try again.')
            Model.update_table1(customer_ID, set)
    elif table == 2:
        seller_ID = "" + input('Attribute to update (where) ') + ""

        View.attribute_list_for_update(2)
        in_restart = True
        while in_restart:
            answer = input('Choose the number of attribute: ')
            if answer == '1':
                value = "" + input('New value of attribute = ') + ""

                set = "name" = {}.format(value)
                in_restart = False
            elif answer == '2':
                value = "" + input('New value of attribute = ') + ""

                set = "surname" = {}.format(value)
                in_restart = False
            else:
                print('Incorrect input, try again.')
                Model.update_table2(seller_ID, set)
    elif table == 3:
        things_barcode = "" + input('Attribute to update (where) barcode = ') + ""

        View.attribute_list_for_update(3)
        in_restart = True
        while in_restart:
            answer = input('Choose the number of attribute: ')
            if answer == '1':
                value = "" + input('New value of attribute = ') + ""

                set = "name" = {}.format(value)
                in_restart = False
            elif answer == '2':
                value = "" + input('New value of attribute = ') + ""

                set = "price" = {}.format(value)
                in_restart = False
            else:
                print('Incorrect input, try again.')
                Model.update_table3(things_barcode, set)
    elif table == 4:
        order_ordernumber = "" + input('Attribute to update (where) ordernumber = ') + ""

        View.attribute_list_for_update(4)
        in_restart = True
        while in_restart:
            answer = input('Choose the number of attribute: ')
            if answer == '1':
                value = "" + input('New value of attribute = ') + ""

                set = "ordertime" = {}.format(value)
                in_restart = False
            else:

```



```

        print('ERROR!\nIncorrect input, try again!')
    except (Exception, psycopg2.Error) as error:
        print('PostgreSQL ERROR!', error)
    incorrect = True
    while incorrect:
        answer = input("Do you want to continue working with the
'select'? \nEnter 1 if yes or 2 if no: ")
        if answer == '2':
            end_select = True
            incorrect = False
        elif answer == '1':
            incorrect = False
        pass
        else:
            print('ERROR!\nPlease, enter your answer!')

    elif choice == '7':
        end_random = False
        while not end_random:
            try:
                table = input("1. Customer\n2. Seller\nChoose the table
number: ")

                table = controller.validate_table(table)
                count = int(input("How many rows do you want to insert: "))
                Model.random(table, count)
                View.show()
            except (Exception, psycopg2.Error) as error:
                print('PostgreSQL ERROR!', error)
                Model.random()
            incorrect = True
            while incorrect:
                answer = input("Do you want to continue working with the
'random data'? \nEnter 1 if yes or 2 if no: ")
                if answer == '2':
                    end_random = True
                    incorrect = False
                elif answer == '1':
                    incorrect = False
                pass
                else:
                    print('ERROR!\nPlease, enter your answer!')

    elif choice == '8':
        break
    else:
        print("Please, choose a number from 1 to 8: ")

```

model.py

```

import controller
import time

```

```

tables = {
    1: 'customer',
    2: 'seller',
    3: 'things',
    4: 'order',
    5: 'order_item',

```

```
}
```

```
class Model:
```

```
    @staticmethod
```

```
    def show_table(table):
```

```
        connection = controller.connection()
```

```
        cursor = connection.cursor()
```

```
        table_name = '""' + tables[table] + '""'
```

```
        print('\nTable ', tables[table])
```

```
        show = f'SELECT * FROM public.{table_name}'
```

```
        print("SQL query: ", show)
```

```
        cursor.execute(show)
```

```
        records = cursor.fetchall()
```

```
        cursor.close()
```

```
        controller.disconnect(connection)
```

```
        return records
```

```
    @staticmethod
```

```
    def insert_table1(customer_ID, customer_name, customer_gender):
```

```
        connection = controller.connection()
```

```
        cursor = connection.cursor()
```

```
        go_on = True
```

```
        while go_on:
```

```
            notice = "This customerID already exists"
```

```
            insert = """DO $$ BEGIN IF NOT EXISTS (SELECT "customerID" FROM
"customer" WHERE "customerID" = {})
```

```
                THEN INSERT INTO "customer"("customerID", "name", "gender")
VALUES ({},{},{}); RAISE NOTICE {}; ELSE RAISE NOTICE {}; END IF; END $$;
```

```
                """.format(customer_ID, customer_ID, customer_name,
customer_gender, "'added'", notice)
```

```
            go_on = False
```

```
            #print("SQL query: ", insert)
```

```
            cursor.execute(insert)
```

```
            connection.commit()
```

```
            print(connection.notices)
```

```
            cursor.close()
```

```
            controller.disconnect(connection)
```

```
    @staticmethod
```

```
    def insert_table2(seller_ID, seller_name, seller_surname):
```

```
        connection = controller.connection()
```

```
        cursor = connection.cursor()
```

```
        go_on = True
```

```
        while go_on:
```

```
            notice = "This sellerID already exists"
```

```
            insert = """DO $$ BEGIN IF NOT EXISTS (SELECT "sellerID" FROM "seller"
WHERE "sellerID" = {})
```

```
                THEN INSERT INTO "seller"("sellerID", "name", "surname")
VALUES ({},{},{}); RAISE NOTICE {}; ELSE RAISE NOTICE {}; END IF; END $$;
```

```
                """.format(seller_ID, seller_ID, seller_name, seller_surname,
"'added'", notice)
```

```
            go_on = False
```

```
            #print("SQL query: ", insert)
```

```
            cursor.execute(insert)
```

```
            connection.commit()
```

```
            print(connection.notices)
```

```
            cursor.close()
```

```
            controller.disconnect(connection)
```

```
    @staticmethod
```



```

    def insert_table3(things_barcode, things_name, things_price,
things_order_itemID):
        connection = controller.connection()
        cursor = connection.cursor()
        go_on = True
        while go_on:
            notice = "'This barcode already exists'"
            insert = """DO $$ BEGIN IF NOT EXISTS (SELECT "order_itemID" FROM
"order_item" WHERE "order_itemID" = {})
                        AND NOT EXISTS (SELECT "barcode" FROM "things" WHERE "barcode"
= {}) THEN
                                INSERT INTO "things"("barcode", "name", "price",
"order_itemID") VALUES ({},{},{},{}); RAISE NOTICE {}; ELSE RAISE NOTICE {}; END
IF; END $$;
                        """.format(things_order_itemID, things_barcode,
things_barcode, things_name, things_price, things_order_itemID, "'added'", notice)
            go_on = False
            print("SQL query: ", insert)
            cursor.execute(insert)
            connection.commit()
            print(connection.notices)
            cursor.close()
            controller.disconnect(connection)

    @staticmethod
    def insert_table4(order_ordernumber, order_ordertime, order_customerID,
order_sellerID):
        connection = controller.connection()
        cursor = connection.cursor()
        go_on = True
        while go_on:
            notice = "'This ordernumber already exists'"
            insert = """DO $$ BEGIN IF NOT EXISTS (SELECT "cusromerID" FROM
"customer" WHERE "customerID" = {})
                        AND NOT EXISTS (SELECT "sellerID" FROM "seller" WHERE
"sellerID" = {}) AND NOT EXISTS (select "ordernumber" from "order" where
"ordernumber" = {}) THEN
                                INSERT INTO "order"("ordernumber", "ordertime", "customerID",
"sellerID") VALUES ({},{},{},{}); RAISE NOTICE {}; ELSE RAISE NOTICE {}; END IF;
                        END $$;""".format(order_customerID, order_sellerID,
order_ordernumber, order_ordernumber, order_ordertime, order_customerID,
order_sellerID, "'added'", notice)
            go_on = False
            print("SQL query: ", insert)
            cursor.execute(insert)
            connection.commit()
            print(connection.notices)
            cursor.close()
            controller.disconnect(connection)

    @staticmethod
    def insert_table5(orderItem_order_itemID, orderItem_count,
orderItem_ordernumber):
        connection = controller.connection()
        cursor = connection.cursor()
        go_on = True
        while go_on:
            notice = "'This order_itemID already exists'"
            insert = """DO $$ BEGIN IF NOT EXISTS (SELECT "ordernumber" FROM
"order" where "ordernumber" = {})

```

```

        AND NOT EXISTS (SELECT "order_itemID" FROM "order_item" WHERE
"order_itemID" = {}) THEN
        INSERT INTO "order_item"("order_itemID", "count",
"ordernumber") VALUES ({},{},{}); 'RAISE NOTICE {}; ELSE RAISE NOTICE {}; END IF;
        END $$;""".format(orderItem_ordernumber,
orderItem_order_itemID, orderItem_order_itemID, orderItem_count,
orderItem_ordernumber, "'added'", notice)
        go_on = False
        print("SQL query: ", insert)
        cursor.execute(insert)
        connection.commit()
        print(connection.notices)
        cursor.close()
        controller.disconnect(connection)

    @staticmethod
    def delete_table1(customer_ID):
        connection = controller.connection()
        cursor = connection.cursor()
        go_on = True
        while go_on:
            delete = f'DELETE FROM "customer" WHERE "customerID" = {customer_ID};'
            \
                f'DELETE FROM "order" WHERE "customerID" = {customer_ID};'
            go_on = False
            print("SQL query: ", delete)
            cursor.execute(delete)
            connection.commit()
            print(connection.notices)
            cursor.close()
            controller.disconnect(connection)

    @staticmethod
    def delete_table2(seller_ID):
        connection = controller.connection()
        cursor = connection.cursor()
        go_on = True
        while go_on:
            delete = f'DELETE FROM "seller" WHERE "sellerID" = {seller_ID};' \
                f'DELETE FROM "order" WHERE "sellerID" = {seller_ID};'
            go_on = False
            print("SQL query: ", delete)
            cursor.execute(delete)
            connection.commit()
            print(connection.notices)
            cursor.close()
            controller.disconnect(connection)

    @staticmethod
    def delete_table3(things_barcode):
        connection = controller.connection()
        cursor = connection.cursor()
        go_on = True
        while go_on:
            delete = f'DELETE FROM "things" WHERE "barcode" = {things_barcode};'
            go_on = False
            print("SQL query: ", delete)
            cursor.execute(delete)
            connection.commit()
            print(connection.notices)
            cursor.close()

```

```

        controller.disconnect(connection)

    @staticmethod
    def delete_table4(order_ordernumber):
        connection = controller.connection()
        cursor = connection.cursor()
        go_on = True
        while go_on:
            delete = f'DELETE FROM "order_item" WHERE "ordernumber" = {order_ordernumber};' \
                    f'DELETE FROM "order" WHERE "ordernumber" = {order_ordernumber};'
            go_on = False
            print("SQL query: ", delete)
            cursor.execute(delete)
            connection.commit()
            print(connection.notices)
            cursor.close()
            controller.disconnect(connection)

    @staticmethod
    def delete_table5(orderItem_order_itemID):
        connection = controller.connection()
        cursor = connection.cursor()
        go_on = True
        while go_on:
            delete = f'DELETE FROM "things" WHERE "order_itemID" = {orderItem_order_itemID};' \
                    f'DELETE FROM "order_item" WHERE "order_itemID" = {orderItem_order_itemID};'
            go_on = False
            print("SQL query: ", delete)
            cursor.execute(delete)
            connection.commit()
            print(connection.notices)
            cursor.close()
            controller.disconnect(connection)

    @staticmethod
    def update_table1(customer_ID, set):
        connection = controller.connection()
        cursor = connection.cursor()
        restart = True
        while restart:
            notice = "'There is nothing to update'"
            update = """DO $$ BEGIN IF EXISTS (SELECT "customerID" FROM "customer"
WHERE "customerID" = {}) THEN
                UPDATE "customer" SET {} WHERE "customerID" = {}; RAISE NOTICE
{}; ELSE RAISE NOTICE {}; END IF;
                END $$;""".format(customer_ID, set, customer_ID, "'updated'",
notice)
            restart = False
            pass
            print("SQL query: ", update)
            cursor.execute(update)
            connection.commit()
            print(connection.notices)
            cursor.close()
            controller.disconnect(connection)
            pass

```

```

@staticmethod
def update_table2(seller_ID, set):
    connection = controller.connection()
    cursor = connection.cursor()
    restart = True
    while restart:
        notice = "'There is nothing to update'"
        update = """DO $$ BEGIN IF EXISTS (SELECT "sellerID" FROM "seller"
WHERE "sellerID" = {}) THEN
            UPDATE "seller" SET {} WHERE "sellerID" = {}; RAISE NOTICE {};
ELSE RAISE NOTICE {}; END IF; END $$;
            """.format(seller_ID, set, seller_ID, "'updated'", notice)
        restart = False
    pass
    print("SQL query: ", update)
    cursor.execute(update)
    connection.commit()
    print(connection.notices)
    cursor.close()
    controller.disconnect(connection)
    pass

@staticmethod
def update_table3(things_barcode, set):
    connection = controller.connection()
    cursor = connection.cursor()
    restart = True
    while restart:
        notice = "'There is nothing to update'"
        update = """DO $$ BEGIN IF EXISTS (SELECT "barcode" FROM "things" WHERE
"barcode" = {})
            THEN UPDATE "things" SET {} WHERE "barcode" = {}; RAISE NOTICE
{}; ELSE RAISE NOTICE {}; END IF; END $$;
            """.format(things_barcode, set, things_barcode, "'updated'",
notice)
        restart = False
    pass
    print("SQL query: ", update)
    cursor.execute(update)
    connection.commit()
    print(connection.notices)
    cursor.close()
    controller.disconnect(connection)
    pass

@staticmethod
def update_table4(order_ordernumber, set):
    connection = controller.connection()
    cursor = connection.cursor()
    restart = True
    while restart:
        notice = "'There is nothing to update'"
        update = """DO $$ BEGIN IF EXISTS (SELECT "ordernumber" FROM "order"
WHERE "ordernumber" = {})
            THEN UPDATE "order" SET {} WHERE "ordernumber" = {}; RAISE
NOTICE {}; ELSE RAISE NOTICE {}; END IF; END $$;
            """.format(order_ordernumber, set, order_ordernumber,
"'updated'", notice)
        restart = False
    pass
    #print("SQL query: ", update)

```

```

        cursor.execute(update)
        connection.commit()
        print(connection.notices)
        cursor.close()
        controller.disconnect(connection)
        pass

    @staticmethod
    def update_table5(orderItem_order_itemID, set):
        connection = controller.connection()
        cursor = connection.cursor()
        restart = True
        while restart:
            notice = "There is nothing to update"
            update = """DO $$ BEGIN IF EXISTS (SELECT "order_itemID" FROM
"order_item" WHERE "order_itemID" = {})
                THEN UPDATE "order_item" SET {} WHERE "order_itemID" = {};
RAISE NOTICE {}; ELSE RAISE NOTICE {}; END IF; END $$;
                """.format(orderItem_order_itemID, set,
orderItem_order_itemID, "updated", notice)
            restart = False
        pass
        #print("SQL query: ", update)
        cursor.execute(update)
        connection.commit()
        print(connection.notices)
        cursor.close()
        controller.disconnect(connection)
        pass

    @staticmethod
    def select_num1(count):
        connection = controller.connection()
        cursor = connection.cursor()
        select = f"""SELECT "count", "name", "price" from (SELECT o."count",
t."name", t."price"
                FROM "things" t LEFT JOIN "order_item" o ON o."order_itemID" =
t."order_itemID"
                WHERE o."count" = '{count}' GROUP BY o."count", t."name",
t."price") AS foo"""
        print("SQL query: ", select)
        beg = int(time.time() * 1000)
        cursor.execute(select)
        end = int(time.time() * 1000) - beg
        records = cursor.fetchall()
        print(f'Time of request = {end} ms')
        cursor.close()
        controller.disconnect(connection)
        return records

    @staticmethod
    def select_num2(customerID):
        connection = controller.connection()
        cursor = connection.cursor()
        select = f"""SELECT "customerID", "ordernumber", "ordertime" FROM (SELECT
c."customerID", o."ordernumber", o."ordertime"
                FROM "order" o LEFT JOIN "customer" c ON c."customerID" =
o."customerID"
                WHERE c."customerID" = '{customerID}' GROUP BY c."customerID",
o."ordernumber", o."ordertime") AS foo"""
        print("SQL query: ", select)

```

```

        beg = int(time.time() * 1000)
        cursor.execute(select)
        end = int(time.time() * 1000) - beg
        records = cursor.fetchall()
        print(f'Time of request = {end} ms')
        cursor.close()
        controller.disconnect(connection)
        return records

    @staticmethod
    def select_num3(sellerID):
        connection = controller.connection()
        cursor = connection.cursor()
        select = f"""SELECT "sellerID", "ordernumber", "ordertime" FROM (SELECT
s."sellerID", o."ordernumber", o."ordertime"
        FROM "order" o LEFT JOIN "seller" s ON s."sellerID" = o."sellerID"
        WHERE s."sellerID" = '{sellerID}' GROUP BY s."sellerID",
o."ordernumber", o."ordertime") AS foo"""
        print("SQL query: ", select)
        beg = int(time.time() * 1000)
        cursor.execute(select)
        end = int(time.time() * 1000) - beg
        records = cursor.fetchall()
        print(f'Time of request = {end} ms')
        cursor.close()
        controller.disconnect(connection)
        return records

    @staticmethod
    def random(table, count):
        connection = controller.connection()
        cursor = connection.cursor()
        go_on = True
        while go_on:
            if table == 1:
                for i in range(count):
                    cursor.execute(f"""INSERT INTO "customer" ("customerID",
"name", "gender")
                                SELECT generate_series as customerID,
                                chr(trunc(65 +
random()*26)::int)||chr(trunc(65 + random()*26)::int)||chr(trunc(65 +
random()*26)::int),
                                (array['F', 'M'])[floor(random() * 2 + 1)]
                                FROM generate_series(5, {count}+5)""")
                    go_on = False
            elif table == 2:
                for i in range(count):
                    cursor.execute(f"""INSERT INTO "seller" ("sellerID", "name",
"surname")
                                SELECT generate_series as sellerID,
                                chr(trunc(65 + random()*26)::int)||chr(trunc(65
+ random()*26)::int),
                                chr(trunc(65 + random()*26)::int)||chr(trunc(65
+ random()*26)::int)||chr(trunc(65 + random()*26)::int)
                                FROM generate_series(5, {count}+5)""")
                    go_on = False
            else:
                print("Please, enter valid number: ")
        print("Insertion of randomized data was successful!")
        connection.commit()

```

```
cursor.close()  
controller.disconnect(connection)
```

Опис функцій модуля “Model”:

show_table – функція, яка виводить таблицю/таблиці;

insert_table1 – функція, яка виконує операцію внесення даних для таблиці 1 (customer);

insert_table2 – функція, яка виконує операцію внесення даних для таблиці 2 (seller);

insert_table3 – функція, яка виконує операцію внесення даних для таблиці 3 (things);

insert_table4 – функція, яка виконує операцію внесення даних для таблиці 4 (order);

insert_table5 – функція, яка виконує операцію внесення даних для таблиці 5 (order_item);

delete_table1 – функція, яка виконує операцію вилучення даних для таблиці 1 (customer);

delete_table2 – функція, яка виконує операцію вилучення даних для таблиці 2 (seller);

delete_table3 – функція, яка виконує операцію вилучення даних для таблиці 3 (things);

delete_table4 – функція, яка виконує операцію вилучення даних для таблиці 4 (order);

delete_table5 – функція, яка виконує операцію вилучення даних для таблиці 5 (order_item);

update_table1 – функція, яка виконує операцію редагування даних для таблиці 1 (customer);

update_table2 – функція, яка виконує операцію редагування для таблиці 2 (seller);

update_table3 – функція, яка виконує операцію редагування даних для таблиці 3 (things);

update_table4 – функція, яка виконує операцію редагування даних для таблиці 4 (order);

update_table5 – функція, яка виконує операцію редагування даних для таблиці 5 (order_item);

select_num1 – функція, яка виконує перший пошуковий запит;

select_num2 – функція, яка виконує другий пошуковий запит;

select_num3 – функція, яка виконує третій пошуковий запит;

random – функція, яка рандомно генерує дані таблиці.