



НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. Ігоря Сікорського»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування та спеціалізованих
комп'ютерних систем**

Розрахунково-графічна робота

з дисципліни
Бази даних і засоби управління

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

Виконала: студентка III курсу

ФПМ групи КВ-12

Ус В. О.

Перевірив:

Павловський В. І.

Київ 2023

Мета роботи: здобуття практичних навичок проектування та побудови реляційних баз даних та створення прикладних програм з базами даних.

Виконання роботи

Обрана предметна галузь передбачає моделювання платформи для замовлення та доставки продуктів харчування.

Сутності

Згідно цієї області для побудови бази даних було виділено наступні сутності:

Користувач: Представляє користувачів платформи. Має атрибути, такі як ID користувача, прізвище та ім'я і номер телефону.

Заклад: Представляє магазини або заклади, які пропонують продукти харчування для замовлення. Має атрибути, такі як ID закладу, назва закладу та номер телефону для замовлення.

Продукт: Описує продукти, доступні для замовлення в закладах. Має атрибути, такі як ID продукту, ID закладу, назва та ціна.

Замовлення: Представляє інформацію про замовлення користувачів. Має такі атрибути, як ID замовлення, ID користувача, ID продуктів та сума для оплати.

Доставка: Відображає інформацію про саму доставку. Має атрибути, такі як ID доставки, ID замовлення, час доставки, адреса доставки.

Опис зв'язків

Зв'язок між "Користувач" і "Замовлення" є 1:N, що означає, що один користувач може мати багато замовлень, але кожне замовлення належить лише одному користувачу. Тобто один користувач може робити багато замовлень, і кожне замовлення належить конкретному користувачеві.

Зв'язок між "Заклад" і "Продукт" є 1:N, що означає, що кожен заклад може пропонувати багато продуктів, але кожен продукт належить лише одному закладу. Цей зв'язок визначає, які продукти доступні в кожному закладі.

Зв'язок між "Замовлення" і "Продукт" є M:N, що означає, що кожне замовлення може містити багато різних продуктів, і в той же час, кожен продукт може бути частиною багатьох різних замовлень. Тобто кілька

замовлень можуть включати один і той самий продукт, і одне замовлення може містити різні продукти.

Зв'язок між "Замовлення" і "Доставка" N:1. Це означає, що кожне замовлення має лише одну доставку, а доставка може мати багато замовлень з різних закладів харчування.

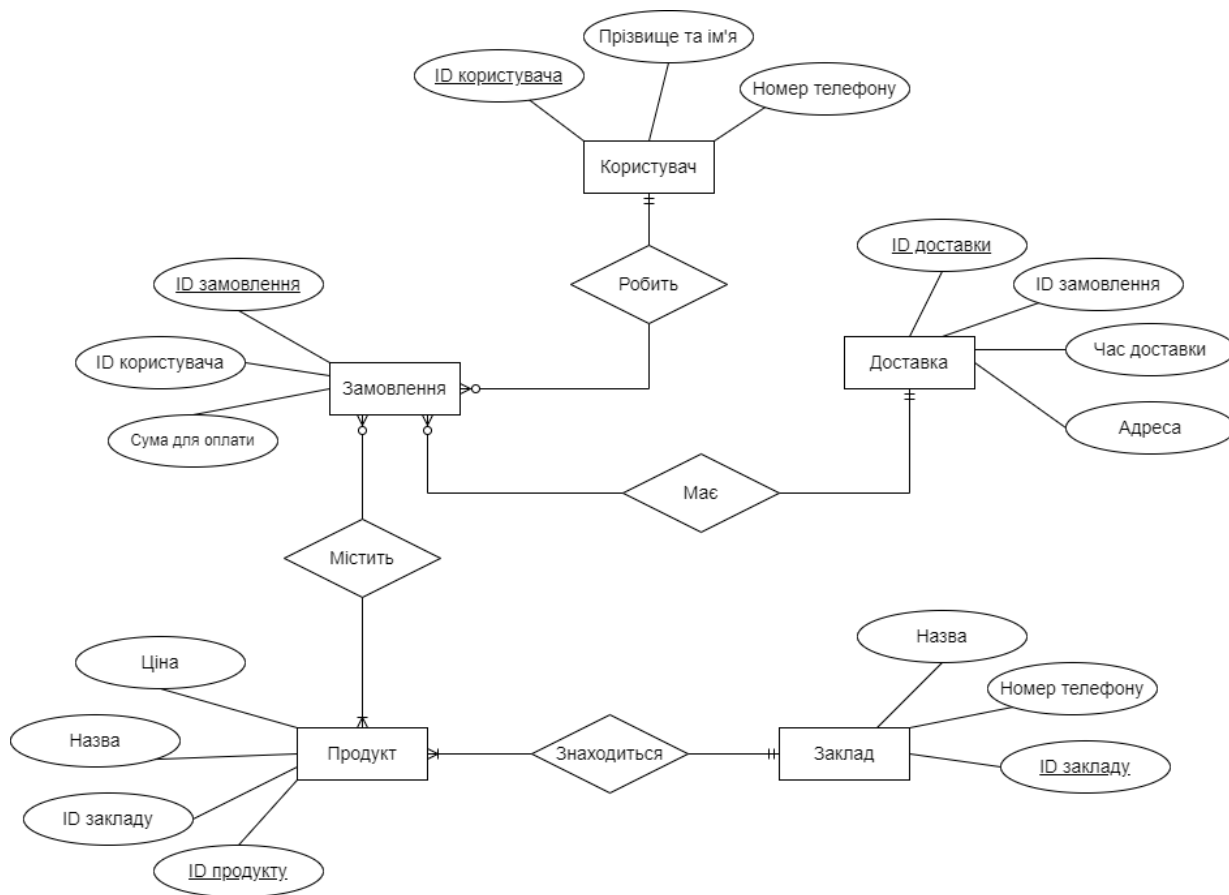


Рисунок 1 - Концептуальна модель даних

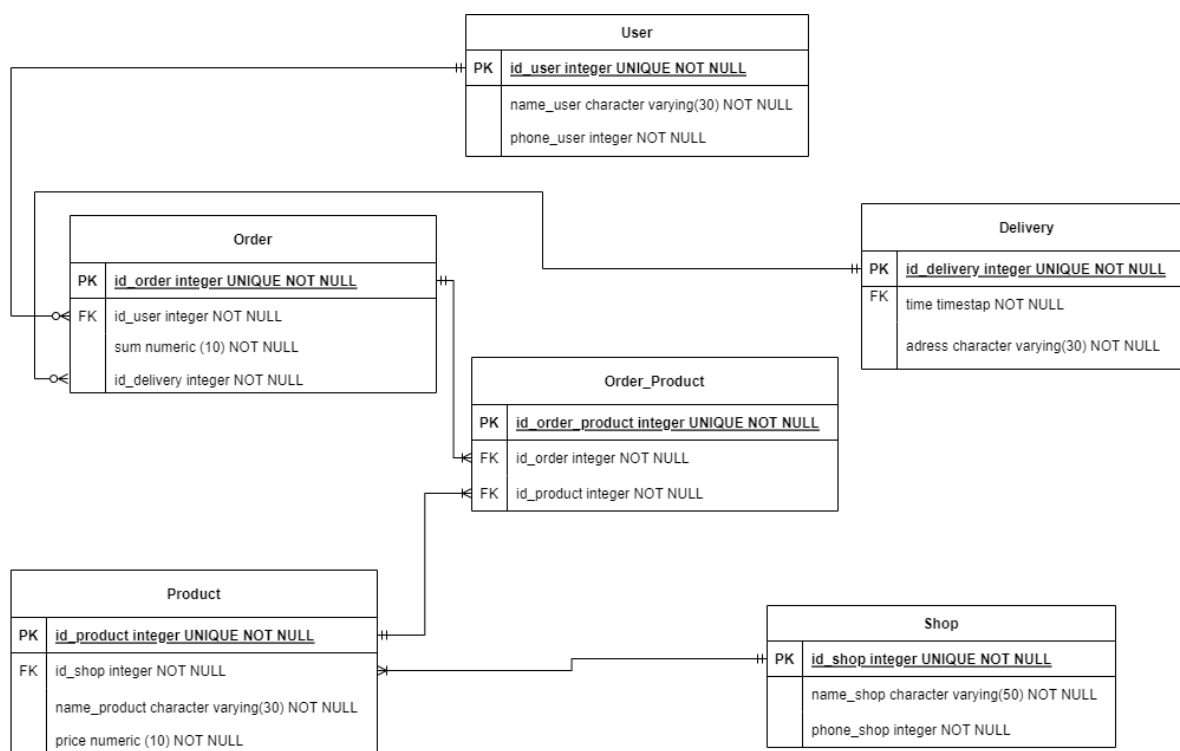


Рисунок 2 - Логічна модель даних

Середовище та компоненти розробки

Для розробки використовувалась мова програмування Python, середовище розробки PyCharm, а також бібліотека psycopg2.

Шаблон проектування

MVC — це шаблон проектування, який застосовується у моїй програмі. В рамках цього шаблону:

- **Model** (Модель): Представляє клас, що визначає логіку обробки та використання даних. Усі компоненти, що відповідають за цю логіку, знаходяться у model.
- **View** (Вид): В даному випадку це консольний інтерфейс, з яким користувач взаємодіє. Компоненти, що представляють цей інтерфейс, відображають необхідні дані у вигляді консольного інтерфейсу.
- **Controller** (Контролер): Представляє клас, який забезпечує взаємодію між користувачем і системою, а також відповідає за обробку та передачу даних. Контролер отримує введені користувачем дані, обробляє їх і, залежно від результатів обробки, повертає користувачеві відповідні висновки, наприклад такі як подання.

Цей підхід дозволяє відділити логіку обробки даних, їх представлення та взаємодії з користувачем, що робить програмний код більш структурованим та забезпечує зміни в окремих компонентах без впливу на інші частини програми.

Структура програми та її опис

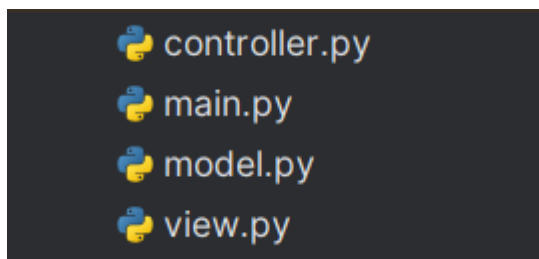


Рисунок 3 - Структура моєї програми

У файлі model.py міститься опис класу моделі, який відповідає за налаштування з'єднання з базою даних та виконання низькорівневих запитів до неї.

У файлі view.py представлений клас, який виводить результати виконання різних дій на екран консолі.

У файлі controller.py визначено інтерфейс взаємодії з користувачем, включаючи обробку запитів на виконання певної дії, виконання пошукових операцій та інші взаємодійні аспекти.

Структура меню програми

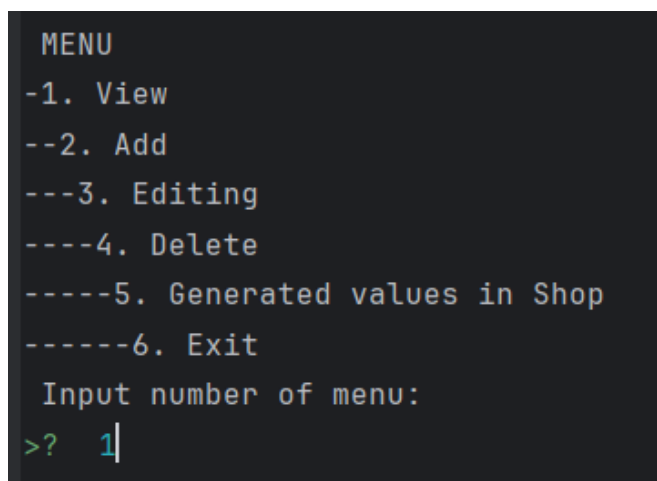
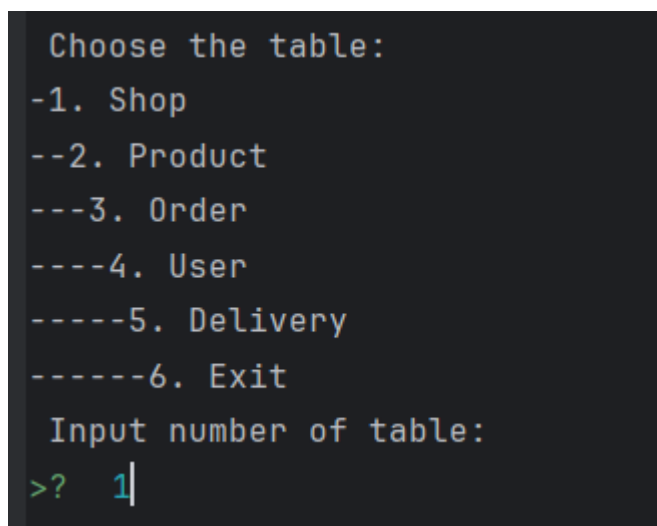


Рисунок 4 - Меню програми

Меню програми складається з 6 пунктів, які користувач може самостійно обрати: пункт 1 – виводить усі рядки обраної таблиці. Пункт 2 – додає елемент в таблиці. Пункт 3 – змінює значення елемента обраної таблиці. Пункт 4 –

видаляє дані з таблиць. Пункт 5 – генерує випадкові значення в таблиці «Shop». Пункт 6 – вихід з меню.

Коли користувач обрав дію над таблицею із запропонованих пунктів меню, йому надається зробити вибір над якою саме таблицею потрібно провести обрану дію, переглянемо на Рисунку 5, наведеному нижче.



```

Choose the table:
-1. Shop
--2. Product
---3. Order
----4. User
-----5. Delivery
-----6. Exit
Input number of table:
>? 1|

```

Рисунок 5 - Вибір таблиці в меню

Фрагмент коду (controller.py), в якому наведено виклик функцій та значення, які їм передаються

```

def generated_shop(self):
    val = self.view.gen_values()
    if val.isdigit():
        self.model.gen_shop(val)
    else:
        print("Error! It must be correct type..")

# Shop
def view_shops(self):
    shops = self.model.get_all_shops()
    self.view.show_shops(shops)
def add_shop(self):
    shop_name = self.view.get_shop_name()
    shop_phone = self.view.get_shop_phone()
    if shop_name.isalpha() and shop_phone.isdigit():
        self.model.add_shop(shop_name, shop_phone)
    else:
        print("Error! It must be correct type..")
def editing_shop(self):
    shop_id = self.view.get_shop_id()
    shop_name = self.view.get_shop_name()
    shop_phone = self.view.get_shop_phone()
    if shop_id.isdigit() and shop_name.isalpha() and
shop_phone.isdigit():
        self.model.editing_shop(shop_id, shop_name, shop_phone)
    else:
        print("Error! It must be correct type..")
def delete_shop(self):

```

```

shop_id = self.view.get_shop_id()
if shop_id.isdigit():
    self.model.delete_shop(shop_id)
else:
    print("Error! It must be correct type..")

# Product
def view_product(self):
    product = self.model.get_all_products()
    self.view.show_products(product)
def add_product(self):
    shop_id, product_name, product_price = self.view.get_product_input()
    if shop_id.isdigit() and product_name.isalpha() and
product_price.isdigit():
        self.model.add_product(shop_id, product_name, product_price)
    else:
        print("Error! It must be correct type..")
def editing_product(self):
    product_id = self.view.get_product_id()
    shop_id, product_name, product_price = self.view.get_product_input()
    if product_id.isdigit() and shop_id.isdigit() and
product_name.isalpha() and product_price.isdigit():
        self.model.editing_product(product_id, shop_id, product_name,
product_price)
    else:
        print("Error! It must be correct type..")
def delete_product(self):
    product_id = self.view.get_product_id()
    if product_id.isdigit():
        self.model.delete_product(product_id)
    else:
        print("Error! It must be correct type..")

# Order
def view_order(self):
    orders = self.model.get_all_order()
    self.view.show_orders(orders)
def add_order(self):
    matrix_product_id = []
    id_user = self.view.get_user_id()
    num_product = self.view.get_num_product()
    id_del = self.view.get_del_id()
    for k in range(int(num_product)):
        matrix_product_id.append(self.view.get_product_id())
    if id_user.isdigit() and num_product.isdigit() and id_del.isdigit():
        self.model.add_order(id_user, matrix_product_id, num_product,
id_del)
    else:
        print("Error! It must be correct type..")
def editing_order(self):
    matrix_product_id = []
    order_id = self.view.get_order_id()
    user_id = self.view.get_user_id()
    num_product = self.view.get_num_product()
    id_del = self.view.get_del_id()
    for k in range(int(num_product)):
        matrix_product_id.append(self.view.get_product_id())
    if order_id.isdigit() and user_id.isdigit() and id_del.isdigit():

```

```

        self.model.editing_order(matrix_product_id, num_product,
order_id, user_id, id_del)
    else:
        print("Error! It must be correct type..")
def delete_order(self):
    order_id = self.view.get_order_id()
    if order_id.isdigit():
        self.model.delete_order(order_id)
    else:
        print("Error! It must be correct type..")

#User
def view_user(self):
    user = self.model.get_all_user()
    self.view.show_user(user)
def add_user(self):
    name, phone = self.view.get_user_input()
    if name.isalpha() and phone.isdigit():
        self.model.add_user(name, phone)
    else:
        print("Error! It must be correct type..")
def editing_user(self):
    user_id = self.view.get_user_id()
    name, phone = self.view.get_user_input()
    if user_id.isdigit() and name.isalpha() and phone.isdigit():
        self.model.editing_user(user_id, name, phone)
    else:
        print("Error! It must be correct type..")
def delete_user(self):
    id_user = self.view.get_user_id()
    if id_user.isdigit():
        self.model.delete_user(id_user)
    else:
        print("Error! It must be correct type..")

# Delivery
def view_del(self):
    delivery = self.model.get_all_delivery()
    self.view.show_del(delivery)
def add_del(self):
    time, adress = self.view.get_del_input()
    self.model.add_del(time, adress)
def editing_del(self):
    id_del = self.view.get_del_id()
    time, adress = self.view.get_del_input()
    if id_del.isdigit():
        self.model.editing_del(id_del, time, adress)
    else:
        print("Error! It must be correct type..")
def delete_del(self):
    id_del = self.view.get_del_id()
    if id_del.isdigit():
        self.model.delete_del(id_del)
    else:
        print("Error! It must be correct type..")

```


Фрагменти коду програм внесення, редагування, вилучення, перегляду та генерації даних у базі даних

Фрагмент коду програми внесення даних

```

def add_shop(self, shop_name, shop_phone):
    c = self.conn.cursor()
    c.execute('INSERT INTO "Shop" ("name_shop", "phone_shop") VALUES (%s,
%s)', (shop_name, shop_phone))
    self.conn.commit()
    print("Shop added successfully!")

def add_product(self, id_shop, name_product, price):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "Shop" WHERE "id_shop" = %s', (id_shop,))
    check1 = c.fetchall()
    if check1:
        c.execute('INSERT INTO "Product" ("id_shop", "name_product",
"price") VALUES (%s, %s, %s)', (id_shop, name_product, price))
        self.conn.commit()
        print("Product added successfully!")
    else:
        print("Error! This identifier id_shop exists")

def add_order(self, id_user, matrix_product_id, num_product, id_del):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "User" WHERE "id_user" = %s', (id_user,))
    user = c.fetchall()
    if user:
        summ = 0
        for k in range(int(num_product)):
            c.execute('SELECT * FROM "Product" WHERE "id_product"=%s', (
matrix_product_id[k]))
            value = c.fetchone()[0]
            summ = summ + value
            c.execute('INSERT INTO "Order" ("id_user", "sum", "id_delivery")
VALUES (%s, %s, %s) RETURNING "id_order"', (id_user, summ, id_del))
            order_id = c.fetchone()[0]
            for m in range(int(num_product)):
                c.execute('INSERT INTO "Order_Product" ("id_order",
"id_product") VALUES (%s, %s)', (order_id, matrix_product_id[m]))
            self.conn.commit()
            print("Order added successfully!")
        else:
            print("Error. not found user")

def add_user(self, name, phone):
    c = self.conn.cursor()
    c.execute('INSERT INTO "User" ("name_user", "phone_user") VALUES (%s,
%s)', (name, phone))
    self.conn.commit()
    print("User added successfully!")

def add_del(self, time_del, adress_del):
    c = self.conn.cursor()
    c.execute('INSERT INTO "Delivery" ("time", "adress") VALUES (%s,
%s)', (time_del, adress_del))
    self.conn.commit()

```

```
print("Delivery added successfully!")
```

Фрагмент коду програми редагування даних

```
def editing_shop(self, shop_id, shop_name, shop_phone):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "Shop" WHERE "id_shop" = %s', (shop_id,))
    check1 = c.fetchall()
    if check1:
        c.execute('UPDATE "Shop" SET "name_shop"=%s, "phone_shop"=%s
WHERE "id_shop"=%s ', (shop_name, shop_phone, shop_id))
        self.conn.commit()
        print("Shop edited successfully!")
    else:
        print("Error! Can't find shop with this id")

def editing_product(self, id_product, id_shop, name_product, price):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "Product" WHERE "id_product" = %s',
(id_shop,))
    check = c.fetchall()
    if check:
        c.execute('UPDATE "Product" SET "id_shop"=%s, "name_product"=%s,
"price"=%s WHERE "id_product"=%s', (id_shop, name_product, price, id_product))
        self.conn.commit()
        print("Product edited successfully!")
    else:
        print("Error! Can't find product with this id")

def editing_order(self, matrix_product_id, num_product, order_id,
user_id, id_del):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "Order" WHERE "id_order" = %s', (order_id,))
    check = c.fetchall()
    if check:
        sum = 0
        for k in range(int(num_product)):
            c.execute('SELECT * FROM "Product" WHERE "id_product"=%s',
(matrix_product_id[k]))
            value = c.fetchone()[0]
            sum = sum + value
        c.execute('UPDATE "Order" SET ("id_user"=%s, "sum"=%s,
"id_delivery"=%s) WHERE "id_order"=%s', (user_id, sum, id_del, order_id))
        self.conn.commit()
        c.execute('DELETE FROM "Order_Product" WHERE "id_order"=%s',
(order_id,))
        self.conn.commit()
        for m in range(int(num_product)):
            c.execute('INSERT INTO "Order_product" ("id_order",
"id_product") VALUES (%s, %s)', (order_id, matrix_product_id[m]))
            self.conn.commit()
        print("Order edited successfully!")
    else:
        print("Error! This id_order not exist")

def editing_user(self, id_user, name, phone):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "User" WHERE "id_user" = %s', (id_user,))
```

```

        check = c.fetchall()
        if check:
            c.execute('UPDATE "User" SET "name_user"=%s, "phone_user"=%s
WHERE "id_user"=%s', (name, phone, id_user))
            self.conn.commit()
            print("User edited successfully!")
        else:
            print("Error! This id_user not exist")

    def editing_del(self, id_del, time, adress):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Delivery" WHERE "id_delivery" = %s',
(id_del,))
        check1 = c.fetchall()
        if check1:
            c.execute('UPDATE "Delivery" SET "time"=%s, "adress"=%s WHERE
"id_delivery"=%s', (time, adress, id_del))
            self.conn.commit()
            print("Delivery deleted successfully!")
        else:
            print("Error! This id_delivery does not exist")

```

Фрагмент коду програми вилучення даних

```

    def delete_shop(self, shop_id):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Shop" WHERE "id_shop" = %s', (shop_id,))
        check1 = c.fetchall()
        c.execute('SELECT * FROM "Product" WHERE "id_shop" = %s', (shop_id,))
        check2 = c.fetchall()
        if check1 and check2:
            c.execute('SELECT "id_product" FROM "Product" WHERE "id_shop" =
%s', (shop_id,))
            n = c.fetchall()
            c.execute('DELETE FROM "Order_Product" WHERE "id_product"= %s',
(n[0],))

            self.conn.commit()
            c.execute('DELETE FROM "Product" WHERE "id_shop"=%s', (shop_id,))
            self.conn.commit()
            c.execute('DELETE FROM "Shop" WHERE "id_shop"=%s', (shop_id,))
            self.conn.commit()
            print("Catalog deleted successfully!")
        elif check1:
            c.execute('DELETE FROM "Shop" WHERE "id_shop"=%s', (shop_id,))
            self.conn.commit()
            print("Shop deleted successfully!")
        else:
            print("Error! This id_shop not exist")

    def delete_product(self, id_product):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Product" WHERE "id_product" = %s',
(id_product,))
        check1 = c.fetchall()
        c.execute('SELECT * FROM "Product" WHERE "id_product" = %s',
(id_product,))
        check2 = c.fetchall()
        if check1 and check2:

```

```

        c.execute('DELETE FROM "Order_Product" WHERE "id_product"=%s',
(id_product,))
        self.conn.commit()
        c.execute('DELETE FROM "Product" WHERE "id_product"=%s',
(id_product,))
        self.conn.commit()
        print("Product deleted successfully!")
    elif check1:
        c.execute('DELETE FROM "Product" WHERE "id_product"=%s',
(id_product,))
        self.conn.commit()
        print("Product deleted successfully!")
    else:
        print("Error! This id_product not exist")

def delete_order(self, id_order):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "Order" WHERE "id_order" = %s', (id_order,))
    check1 = c.fetchall()
    c.execute('SELECT * FROM "Order_Product" WHERE "id_order" = %s',
(id_order,))
    check2 = c.fetchall()
    if check1 and check2:
        c.execute('DELETE FROM "Order_Product" WHERE "id_order"=%s',
(id_order,))
        self.conn.commit()
        c.execute('DELETE FROM "Order" WHERE "id_order"=%s', (id_order,))
        self.conn.commit()
        print("Order deleted successfully!")
    elif check1:
        c.execute('DELETE FROM "Order" WHERE "id_order"=%s', (id_order,))
        self.conn.commit()
        print("Order deleted successfully!")
    else:
        print("Error! This id_order not exist")

def delete_user(self, id_user):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "User" WHERE "id_user" = %s', (id_user,))
    check1 = c.fetchall()
    if check1:
        c.execute('SELECT "id_order" FROM "Order" WHERE "id_user"=%s',
(id_user,))
        order_ids = c.fetchall()
        for order_id in order_ids:
            c.execute('DELETE FROM "Order_Product" WHERE "id_order"=%s',
(order_id[0],))
            self.conn.commit()
        c.execute('DELETE FROM "Order" WHERE "id_user"=%s', (id_user,))
        self.conn.commit()
        c.execute('DELETE FROM "User" WHERE "id_user"=%s', (id_user,))
        self.conn.commit()
        print("User deleted successfully!")
    else:
        print("Error! This id_user does not exist")

def delete_del(self, id_del):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "Delivery" WHERE "id_delivery" = %s',
(id_del,))

```

```

        check1 = c.fetchall()
        if check1:
            c.execute('SELECT "id_order" FROM "Order" WHERE
"id_delivery"=%s', (id_del,))
            order_ids = c.fetchall()
            for order_id in order_ids:
                c.execute('DELETE FROM "Order_Product" WHERE "id_order"=%s',
(order_id[0],))
            c.execute('DELETE FROM "Order" WHERE "id_delivery"=%s',
(id_del,))
            c.execute('DELETE FROM "Delivery" WHERE "id_delivery"=%s',
(id_del,))
            self.conn.commit()
            print("Delivery deleted successfully!")
        else:
            print("Error! This id_delivery does not exist")

```

Фрагмент коду програми перегляду даних

```

def get_all_shops(self):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "Shop"')
    return c.fetchall()

def get_all_products(self):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "Product"')
    return c.fetchall()

def get_all_order(self):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "Order"')
    return c.fetchall()

def get_all_user(self):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "User"')
    return c.fetchall()

def get_all_delivery(self):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "Delivery"')
    return c.fetchall()

```

Фрагмент коду програми генерації випадкових значень в таблиці «Shop»

```

def gen_shop(self, val):
    c = self.conn.cursor()
    for i in range(val):
        name_shop = ''.join(random.choices(string.ascii_letters, k=6))
        phone_shop = ''.join(random.choices(string.digits, k=8))
        c.execute('INSERT INTO "Shop" ("name_shop", "phone_shop") VALUES
(%s, %s)', (name_shop, phone_shop))
    self.conn.commit()

```

```
print("Value in Shop is generated!")
```

Повний текст програми

controller.py

```
from model import Model
from view import View

class Controller:
    def __init__(self):
        self.model = Model()
        self.view = View()

    def run(self):
        while True:
            item = self.show_menu()
            if item == '1':
                self.views()
            elif item == '2':
                self.add()
            elif item == '3':
                self.editing()
            elif item == '4':
                self.delete()
            elif item == '5':
                self.generated_shop()
            elif item == '6':
                break

    def show_menu(self):
        self.view.show_message("\n MENU")
        self.view.show_message("-1. View")
        self.view.show_message("--2. Add")
        self.view.show_message("---3. Editing")
        self.view.show_message("----4. Delete")
        self.view.show_message("-----5. Generated values in Shop")
        self.view.show_message("-----6. Exit")
        return input(" Input number of menu: ")

    def menu(self):
        self.view.show_message("\n Choose the table:")
        self.view.show_message("-1. Shop")
        self.view.show_message("--2. Product")
        self.view.show_message("---3. Order")
        self.view.show_message("----4. User")
        self.view.show_message("-----5. Delivery")
        self.view.show_message("-----6. Exit")
        return input(" Input number of table: ")

    def views(self):
        self.view.show_message("\nView:")
        while True:
            item = self.menu()
            if item == '1':
                self.view_shops()
            elif item == '2':
                self.view_product()
            elif item == '3':
```

```

        self.view_order()
    elif item == '4':
        self.view_user()
    elif item == '5':
        self.view_del()
    elif item == '6':
        break
def add(self):
    self.view.show_message("\nAdd:")
    while True:
        item1 = self.menu()
        if item1 == '1':
            self.add_shop()
        elif item1 == '2':
            self.add_product()
        elif item1 == '3':
            self.add_order()
        elif item1 == '4':
            self.add_user()
        elif item1 == '5':
            self.add_del()
        elif item1 == '6':
            break

def editing(self):
    self.view.show_message("\nEdit:")
    while True:
        item2 = self.menu()
        if item2 == '1':
            self.editing_shop()
        elif item2 == '2':
            self.editing_product()
        elif item2 == '3':
            self.editing_order()
        elif item2 == '4':
            self.editing_user()
        elif item2 == '5':
            self.editing_del()
        elif item2 == '6':
            break

def delete(self):
    self.view.show_message("\nDelete:")
    while True:
        item3 = self.menu()
        if item3 == '1':
            self.delete_shop()
        elif item3 == '2':
            self.delete_product()
        elif item3 == '3':
            self.delete_order()
        elif item3 == '4':
            self.delete_user()
        elif item3 == '5':
            self.delete_del()
        elif item3 == '6':
            break

def generated_shop(self):
    val = self.view.gen_values()
    if val.isdigit():
        self.model.gen_shop(val)
    else:

```

```

        print("Error! It must be correct type..")

# Shop
def view_shops(self):
    shops = self.model.get_all_shops()
    self.view.show_shops(shops)
def add_shop(self):
    shop_name = self.view.get_shop_name()
    shop_phone = self.view.get_shop_phone()
    if shop_name.isalpha() and shop_phone.isdigit():
        self.model.add_shop(shop_name, shop_phone)
    else:
        print("Error! It must be correct type..")
def editing_shop(self):
    shop_id = self.view.get_shop_id()
    shop_name = self.view.get_shop_name()
    shop_phone = self.view.get_shop_phone()
    if shop_id.isdigit() and shop_name.isalpha() and
shop_phone.isdigit():
        self.model.editing_shop(shop_id, shop_name, shop_phone)
    else:
        print("Error! It must be correct type..")
def delete_shop(self):
    shop_id = self.view.get_shop_id()
    if shop_id.isdigit():
        self.model.delete_shop(shop_id)
    else:
        print("Error! It must be correct type..")

# Product
def view_product(self):
    product = self.model.get_all_products()
    self.view.show_products(product)
def add_product(self):
    shop_id, product_name, product_price = self.view.get_product_input()
    if shop_id.isdigit() and product_name.isalpha() and
product_price.isdigit():
        self.model.add_product(shop_id, product_name, product_price)
    else:
        print("Error! It must be correct type..")
def editing_product(self):
    product_id = self.view.get_product_id()
    shop_id, product_name, product_price = self.view.get_product_input()
    if product_id.isdigit() and shop_id.isdigit() and
product_name.isalpha() and product_price.isdigit():
        self.model.editing_product(product_id, shop_id, product_name,
product_price)
    else:
        print("Error! It must be correct type..")
def delete_product(self):
    product_id = self.view.get_product_id()
    if product_id.isdigit():
        self.model.delete_product(product_id)
    else:
        print("Error! It must be correct type..")

# Order
def view_order(self):

```



```

orders = self.model.get_all_order()
self.view.show_orders(orders)
def add_order(self):
    matrix_product_id = []
    id_user = self.view.get_user_id()
    num_product = self.view.get_num_product()
    id_del = self.view.get_del_id()
    for k in range(int(num_product)):
        matrix_product_id.append(self.view.get_product_id())
    if id_user.isdigit() and num_product.isdigit() and id_del.isdigit():
        self.model.add_order(id_user, matrix_product_id, num_product,
id_del)
    else:
        print("Error! It must be correct type..")
def editing_order(self):
    matrix_product_id = []
    order_id = self.view.get_order_id()
    user_id = self.view.get_user_id()
    num_product = self.view.get_num_product()
    id_del = self.view.get_del_id()
    for k in range(int(num_product)):
        matrix_product_id.append(self.view.get_product_id())
    if order_id.isdigit() and user_id.isdigit() and id_del.isdigit():
        self.model.editing_order(matrix_product_id, num_product,
order_id, user_id, id_del)
    else:
        print("Error! It must be correct type..")
def delete_order(self):
    order_id = self.view.get_order_id()
    if order_id.isdigit():
        self.model.delete_order(order_id)
    else:
        print("Error! It must be correct type..")

#User
def view_user(self):
    user = self.model.get_all_user()
    self.view.show_user(user)
def add_user(self):
    name, phone = self.view.get_user_input()
    if name.isalpha() and phone.isdigit():
        self.model.add_user(name, phone)
    else:
        print("Error! It must be correct type..")
def editing_user(self):
    user_id = self.view.get_user_id()
    name, phone = self.view.get_user_input()
    if user_id.isdigit() and name.isalpha() and phone.isdigit():
        self.model.editing_user(user_id, name, phone)
    else:
        print("Error! It must be correct type..")
def delete_user(self):
    id_user = self.view.get_user_id()
    if id_user.isdigit():
        self.model.delete_user(id_user)
    else:
        print("Error! It must be correct type..")

# Delivery

```

```

def view_del(self):
    delivery = self.model.get_all_delivery()
    self.view.show_del(delivery)
def add_del(self):
    time, adress = self.view.get_del_input()
    self.model.add_del(time, adress)
def editing_del(self):
    id_del = self.view.get_del_id()
    time, adress = self.view.get_del_input()
    if id_del.isdigit():
        self.model.editing_del(id_del, time, adress)
    else:
        print("Error! It must be correct type..")
def delete_del(self):
    id_del = self.view.get_del_id()
    if id_del.isdigit():
        self.model.delete_del(id_del)
    else:
        print("Error! It must be correct type..")

```

view.py

```

class View:
    def show_message(self, message):
        print(message)

    #Shop
    def show_shops(self, Shop):
        if Shop:
            print("Shops:")
            for shops in Shop:
                print(f"Shop id: {shops[0]}, name shop: {shops[1]}, phone
shop: {shops[2]}")
            else:
                print("Not found!")

    def get_shop_name(self):
        shop_name = input("Input shop name: ")
        return shop_name

    def get_shop_id(self):
        shop_id = input("Input shop id: ")
        return shop_id

    def get_shop_phone(self):
        shop_phone = input("Input shop number phone: ")
        return shop_phone

    #Product
    def show_products(self, product):
        if product:
            print("Products:")
            for p in product:
                print(f"id_product: {p[0]}, id_shop: {p[1]}, name_product:
{p[2]}, price: {p[3]}")
            else:
                print("Not found")

    def get_product_input(self):

```

```

shop_id = input("Input shop id: ")
product_name = input("Input product name: ")
product_price = input("Input product price: ")
return shop_id, product_name, product_price

def get_product_id(self):
    product_id = input("Input product id: ")
    return product_id

#Order
def show_orders(self, Order):
    if Order:
        print("Shops:")
        for o in Order:
            print(f"id_order: {o[0]}, id_user: {o[1]}, sum: {o[2]},
id_delivery: {o[3]}")
    else:
        print("No order found.")

def get_num_product(self):
    num = input("Input number of products: ")
    return num

def get_order_id(self):
    order_id = input("Input id order: ")
    return order_id

# user
def show_user(self, user):
    if user:
        print("Users:")
        for U in user:
            print(f"id: {U[0]}, name: {U[1]}, phone: {U[2]}")
    else:
        print("Not found")

def get_user_input(self):
    name = input("Input name user: ")
    phone = input("Input phone user: ")
    return name, phone

def get_user_id(self):
    user_id = input("Input user id: ")
    return user_id

#Delivery
def show_del(self, delivery):
    if delivery:
        print("Delivery:")
        for D in delivery:
            print(f"id: {D[0]}, time: {D[1]}, adress: {D[2]}")
    else:
        print("Not found")

def get_del_input(self):
    time = input("Input time delivery: ")
    adress_del = input("Input adress delivery: ")
    return time, adress_del

def get_del_id(self):
    id_del = input("Input delivery id: ")
    return id_del

```

```
def gen_values(self):
    val = int(input("Input number of generated values: "))
    return val
```

model.py

```
import psycopg2
import random
import string
class Model:
    def __init__(self):
        self.conn = psycopg2.connect(
            dbname='lab1',
            user='postgres',
            password='0000',
            host='localhost',
            port=5432
        )

    def gen_shop(self, val):
        c = self.conn.cursor()
        for i in range(val):
            name_shop = ''.join(random.choices(string.ascii_letters, k=6))
            phone_shop = ''.join(random.choices(string.digits, k=8))
            c.execute('INSERT INTO "Shop" ("name_shop", "phone_shop") VALUES (%s, %s)', (name_shop, phone_shop))
            self.conn.commit()
            print("Value in Shop is generated!")

        # Shop
    def add_shop(self, shop_name, shop_phone):
        c = self.conn.cursor()
        c.execute('INSERT INTO "Shop" ("name_shop", "phone_shop") VALUES (%s, %s)', (shop_name, shop_phone))
        self.conn.commit()
        print("Shop added successfully!")

    def editing_shop(self, shop_id, shop_name, shop_phone):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Shop" WHERE "id_shop" = %s', (shop_id,))
        check1 = c.fetchall()
        if check1:
            c.execute('UPDATE "Shop" SET "name_shop"=%s, "phone_shop"=%s WHERE "id_shop"=%s ', (shop_name, shop_phone, shop_id))
            self.conn.commit()
            print("Shop edited successfully!")
        else:
            print("Error! Can't find shop with this id")

    def delete_shop(self, shop_id):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Shop" WHERE "id_shop" = %s', (shop_id,))
        check1 = c.fetchall()
        c.execute('SELECT * FROM "Product" WHERE "id_shop" = %s', (shop_id,))
        check2 = c.fetchall()
        if check1 and check2:
            c.execute('SELECT "id_product" FROM "Product" WHERE "id_shop" = %s', (shop_id,))
            n = c.fetchall()
```

```

        c.execute('DELETE FROM "Order_Product" WHERE "id_product"= %s',
(n[0],))
        self.conn.commit()
        c.execute('DELETE FROM "Product" WHERE "id_shop"=%s', (shop_id,))
        self.conn.commit()
        c.execute('DELETE FROM "Shop" WHERE "id_shop"=%s', (shop_id,))
        self.conn.commit()
        print("Catalog deleted successfully!")
    elif check1:
        c.execute('DELETE FROM "Shop" WHERE "id_shop"=%s', (shop_id,))
        self.conn.commit()
        print("Shop deleted successfully!")
    else:
        print("Error! This id_shop not exist")

def get_all_shops(self):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "Shop"')
    return c.fetchall()

#Product
def add_product(self, id_shop, name_product, price):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "Shop" WHERE "id_shop" = %s', (id_shop,))
    check1 = c.fetchall()
    if check1:
        c.execute('INSERT INTO "Product" ("id_shop", "name_product",
"price") VALUES (%s, %s, %s)', (id_shop, name_product, price))
        self.conn.commit()
        print("Product added successfully!")
    else:
        print("Error! This identifier id_shop exists")

def editing_product(self, id_product, id_shop, name_product, price):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "Product" WHERE "id_product" = %s',
(id_shop,))
    check = c.fetchall()
    if check:
        c.execute('UPDATE "Product" SET "id_shop"=%s, "name_product"=%s,
"price"=%s WHERE "id_product"=%s', (id_shop, name_product, price, id_product))
        self.conn.commit()
        print("Product edited successfully!")
    else:
        print("Error! Can't find product with this id")

def delete_product(self, id_product):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "Product" WHERE "id_product" = %s',
(id_product,))
    check1 = c.fetchall()
    c.execute('SELECT * FROM "Product" WHERE "id_product" = %s',
(id_product,))
    check2 = c.fetchall()
    if check1 and check2:
        c.execute('DELETE FROM "Order_Product" WHERE "id_product"=%s',
(id_product,))
        self.conn.commit()
        c.execute('DELETE FROM "Product" WHERE "id_product"=%s',
(id_product,))
        self.conn.commit()
        print("Product deleted successfully!")

```

```

        elif check1:
            c.execute('DELETE FROM "Product" WHERE "id_product"=%s',
(id_product,))
            self.conn.commit()
            print("Product deleted successfully!")
        else:
            print("Error! This id_product not exist")

    def get_all_products(self):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Product"')
        return c.fetchall()

# Order
    def add_order(self, id_user, matrix_product_id, num_product, id_del):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "User" WHERE "id_user" = %s', (id_user,))
        user = c.fetchall()
        if user:
            summ = 0
            for k in range(int(num_product)):
                c.execute('SELECT * FROM "Product" WHERE "id_product"=%s', (
matrix_product_id[k]))
                value = c.fetchone()[0]
                summ = summ + value
            c.execute('INSERT INTO "Order" ("id_user", "sum", "id_delivery")
VALUES (%s, %s, %s) RETURNING "id_order"', (id_user, summ, id_del))
            order_id = c.fetchone()[0]
            for m in range(int(num_product)):
                c.execute('INSERT INTO "Order_Product" ("id_order",
"id_product") VALUES (%s, %s)', (order_id, matrix_product_id[m]))
            self.conn.commit()
            print("Order added successfully!")
        else:
            print("Error. not found user")

    def editing_order(self, matrix_product_id, num_product, order_id,
user_id, id_del):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Order" WHERE "id_order" = %s', (order_id,))
        check = c.fetchall()
        if check:
            sum = 0
            for k in range(int(num_product)):
                c.execute('SELECT * FROM "Product" WHERE "id_product"=%s',
(matrix_product_id[k]))
                value = c.fetchone()[0]
                sum = sum + value
            c.execute('UPDATE "Order" SET ("id_user"=%s, "sum"=%s,
"id_delivery"=%s) WHERE "id_order"=%s', (user_id, sum, id_del, order_id))
            self.conn.commit()
            c.execute('DELETE FROM "Order_Product" WHERE "id_order"=%s',
(order_id,))
            self.conn.commit()
            for m in range(int(num_product)):
                c.execute('INSERT INTO "Order_product" ("id_order",
"id_product") VALUES (%s, %s)', (order_id, matrix_product_id[m]))
                self.conn.commit()
            print("Order edited successfully!")
        else:
            print("Error! This id_order not exist")

```

```

def delete_order(self, id_order):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "Order" WHERE "id_order" = %s', (id_order,))
    check1 = c.fetchall()
    c.execute('SELECT * FROM "Order_Product" WHERE "id_order" = %s',
(id_order,))
    check2 = c.fetchall()
    if check1 and check2:
        c.execute('DELETE FROM "Order_Product" WHERE "id_order"=%s',
(id_order,))
        self.conn.commit()
        c.execute('DELETE FROM "Order" WHERE "id_order"=%s', (id_order,))
        self.conn.commit()
        print("Order deleted successfully!")
    elif check1:
        c.execute('DELETE FROM "Order" WHERE "id_order"=%s', (id_order,))
        self.conn.commit()
        print("Order deleted successfully!")
    else:
        print("Error! This id_order not exist")

def get_all_order(self):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "Order"')
    return c.fetchall()

# User
def add_user(self, name, phone):
    c = self.conn.cursor()
    c.execute('INSERT INTO "User" ("name_user", "phone_user") VALUES (%s,
%s)', (name, phone))
    self.conn.commit()
    print("User added successfully!")

def get_all_user(self):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "User"')
    return c.fetchall()

def editing_user(self, id_user, name, phone):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "User" WHERE "id_user" = %s', (id_user,))
    check = c.fetchall()
    if check:
        c.execute('UPDATE "User" SET "name_user"=%s, "phone_user"=%s
WHERE "id_user"=%s', (name, phone, id_user))
        self.conn.commit()
        print("User edited successfully!")
    else:
        print("Error! This id_user not exist")

def delete_user(self, id_user):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "User" WHERE "id_user" = %s', (id_user,))
    check1 = c.fetchall()
    if check1:
        c.execute('SELECT "id_order" FROM "Order" WHERE "id_user"=%s',
(id_user,))
        order_ids = c.fetchall()
        for order_id in order_ids:
            c.execute('DELETE FROM "Order_Product" WHERE "id_order"=%s',
(order_id[0],))
        self.conn.commit()

```

```

        c.execute('DELETE FROM "Order" WHERE "id_user"=%s', (id_user,))
        self.conn.commit()
        c.execute('DELETE FROM "User" WHERE "id_user"=%s', (id_user,))
        self.conn.commit()
        print("User deleted successfully!")
    else:
        print("Error! This id_user does not exist")

# Delivery
def add_del(self, time_del, adress_del):
    c = self.conn.cursor()
    c.execute('INSERT INTO "Delivery" ("time", "adress") VALUES (%s,
%s)', (time_del, adress_del))
    self.conn.commit()
    print("Delivery added successfully!")

def get_all_delivery(self):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "Delivery"')
    return c.fetchall()

def editing_del(self, id_del, time, adress):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "Delivery" WHERE "id_delivery" = %s',
(id_del,))
    check1 = c.fetchall()
    if check1:
        c.execute('UPDATE "Delivery" SET "time"=%s, "adress"=%s WHERE
"id_delivery"=%s', (time, adress, id_del))
        self.conn.commit()
        print("Delivery deleted successfully!")
    else:
        print("Error! This id_delivery does not exist")

def delete_del(self, id_del):
    c = self.conn.cursor()
    c.execute('SELECT * FROM "Delivery" WHERE "id_delivery" = %s',
(id_del,))
    check1 = c.fetchall()
    if check1:
        c.execute('SELECT "id_order" FROM "Order" WHERE
"id_delivery"=%s', (id_del,))
        order_ids = c.fetchall()
        for order_id in order_ids:
            c.execute('DELETE FROM "Order_Product" WHERE "id_order"=%s',
(order_id[0],))
        c.execute('DELETE FROM "Order" WHERE "id_delivery"=%s',
(id_del,))
        c.execute('DELETE FROM "Delivery" WHERE "id_delivery"=%s',
(id_del,))
        self.conn.commit()
        print("Delivery deleted successfully!")
    else:
        print("Error! This id_delivery does not exist")

```

main.py

```

from controller import Controller

if __name__ == "__main__":

```



```
controller = Controller()  
controller.run()
```

Скріншоти результатів виконання операції вставки запису в дочірню таблицю

	id_product [PK] integer	id_shop integer	name_product character varying (30)	price numeric (10,2)
1	6	5	Carrot	36.00
2	7	6	Cucumber	120.00
3	8	7	Rice	83.00

```





-1. Shop
--2. Product
---3. Order
----4. User
-----5. Delivery
-----6. Exit

Input number of table: >? 2
Input shop id: >? 8
Input product name: >? Banana
Input product price: >? 77
Product added successfully!

```

	id_product [PK] integer	id_shop integer	name_product character varying (30)	price numeric (10,2)
1	6	5	Carrot	36.00
2	7	6	Cucumber	120.00
3	8	7	Rice	83.00
4	10	8	Banana	77.00

Скріншоти результатів виконання операції редагування таблиці

	id_product [PK] integer 	id_shop integer 	name_product character varying (30) 	price numeric (10,2) 
1	6	5	Carrot	36.00
2	7	6	Cucumber	120.00
3	8	7	Rice	83.00
4	10	8	Banana	77.00

Edit:

Choose the table:

- 1. Shop
- 2. Product
- 3. Order
- 4. User
- 5. Delivery
- 6. Exit

Input number of table: >? 2





Input product id: >? 6

Input shop id: >? 9

Input product name: >? Pizza

Input product price: >? 125

Product edited successfully!

	id_product [PK] integer 	id_shop integer 	name_product character varying (30) 	price numeric (10,2) 
1	6	9	Pizza	125.00
2	7	6	Cucumber	120.00
3	8	7	Rice	83.00
4	10	8	Banana	77.00

Скріншоти результатів виконання операції видалення

	id_product [PK] integer	id_shop integer	name_product character varying (30)	price numeric (10,2)
1	6	9	Pizza	125.00
2	7	6	Cucumber	120.00
3	8	7	Rice	83.00
4	10	8	Banana	77.00

Delete:

Choose the table:

-1. Shop

--2. Product

---3. Order

----4. User

-----5. Delivery

-----6. Exit





Input number of table: >? 2

Input product id: >? 8

Product deleted successfully!

	id_product [PK] integer	id_shop integer	name_product character varying (30)	price numeric (10,2)
1	6	9	Pizza	125.00
2	7	6	Cucumber	120.00
3	10	8	Banana	77.00

Скріншоти результатів виконання перегляду

	id_product [PK] integer 	id_shop integer 	name_product character varying (30) 	price numeric (10,2) 
1	6	9	Pizza	125.00
2	7	6	Cucumber	120.00
3	10	8	Banana	77.00

View:

Choose the table:

- 1. Shop
- 2. Product
- 3. Order
- 4. User
- 5. Delivery
- 6. Exit

Input number of table: >? 2

Products:

id_product: 7, id_shop: 6, name_product: Cucumber, price: 120.00
 id_product: 10, id_shop: 8, name_product: Banana, price: 77.00
 id_product: 6, id_shop: 9, name_product: Pizza, price: 125.00

Скріншоти результатів виконання операції генерування

	id_shop [PK] integer	name_shop character varying (50)	phone_shop integer
1	5	Silpo	45565667
2	6	Kolosok	54667767
3	7	Torba	45748849
4	8	Wolmart	45677887
5	9	uvAbHr	913393067
6	10	udWSwr	2661283

```

MENU
-1. View
--2. Add
---3. Editing
----4. Delete
-----5. Generated values in Shop
-----6. Exit
Input number of menu: >? 5
Input number of generated values: >? 4
Value in Shop is generated!

```

	id_shop [PK] integer	name_shop character varying (50)	phone_shop integer
1	5	Silpo	45565667
2	6	Kolosok	54667767
3	7	Torba	45748849
4	8	Wolmart	45677887
5	9	uvAbHr	913393067
6	10	udWSwr	2661283
7	11	bvybjw	49005444
8	12	MkUJbE	74704460
9	13	GKwvPf	958984
10	14	ldLBTg	30793661

Контакты:

GitHub: <https://github.com/vicka-us/DataBase>

Telegram: @vicka_us