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»

Виконав: студент III курсу

ФПМ групи КВ-12

Петрушин В.Б.

Перевірив:

Павловський В.І.

Київ 2023

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

Хід виконання роботи

«Система управління лояльністю клієнтів в роздрібній торгівлі»

Сутності

Згідно даної теми, для побудови бази даних було виділено наступні сутності:

Клієнт з атрибутами: ПІБ клієнта, номер телефону, id клієнта, стать. Призначення: збереження даних щодо клієнтів.

Товар з атрибутами: назва товару, id товару, опис, ціна. Призначення: збереження даних, які стосуються товарів.

Замовлення з атрибутами: дата замовлення, id замовлення, id клієнта, вартість замовлення. Призначення: збереження даних, що стосуються деталей замовлень.

Програма лояльності з атрибутами: id програми, id клієнта, сумарна вартість всіх замовлень, рівень знижки (1 рівень – 5%, від 1 500 грн., 2 – 10%, від 5 000 грн., 3 – 15%, від 10 000 грн.). Призначення: збереження інформації щодо програми лояльності.

Так як лояльність означає прихильність клієнтів до певної компанії (магазину, тощо), за допомогою даної системи користувач може керувати лояльністю в своєму конкретному бізнесі (компанії, магазині, тощо). Тому сутності магазин, чи компанія немає.

Опис зв’язків

Кожен клієнт може взагалі не мати жодних замовлень, або мати багато замовлень, але кожне замовлення обов’язково повинне належати одному конкретному клієнту. Тому між сутностями Клієнт і Замовлення існує зв’язок 1:N.

Кожне замовлення може містити багато товарів, а кожен товар може бути частиною багатьох замовлень. Кожне замовлення повинне містити принаймні один товар. Тому між сутностями Замовлення і Товар існує зв’язок M:N.

Кожен клієнт може мати програму лояльності, але так як програма лояльності одна, то кожен клієнт може мати лише одну програму лояльності. Тому між сутностями Клієнт і Програма лояльності існує зв’язок N:1.

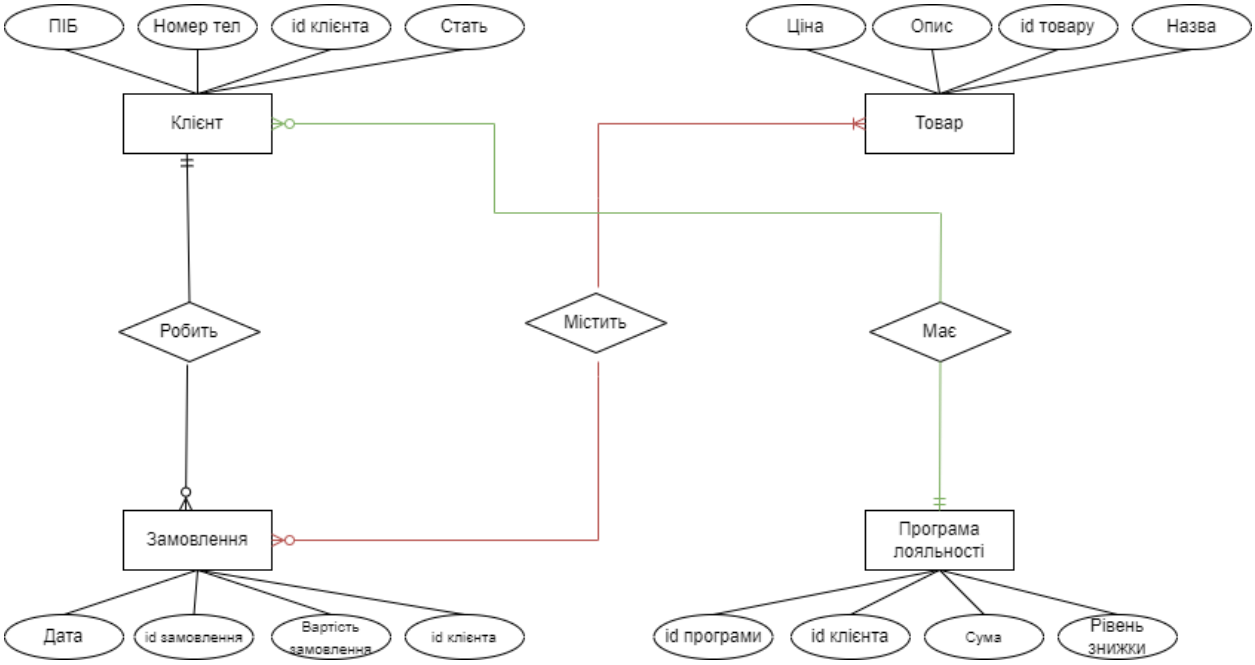

Рисунок 1 – Концептуальна модель даних

Опис процесу перетворення:

Сутність Клієнт ми перетворили в таблицю Client, з первинним ключем (ідентифікатором) client_id та атрибутами: client_name, client_ph_number, client_sex.

Сутність Товар ми перетворили в таблицю Goods, з первинним ключем (ідентифікатором) goods_id та атрибутами: goods_name, goods_description, goods_price.

Сутність Програма лояльності ми перетворили в таблицю Loyalty_program, з первинним ключем (ідентифікатором) program_id та атрибутами: client_id, loyalty_program_sum, discount_level.

Сутність Замовлення ми перетворили в таблицю Order, з первинним ключем (ідентифікатором) order_id та атрибутами: order_date, client_id, order_price.

Зв’язок M:N між сутностями Товар та Замовлення зумовив створення додаткової таблиці Order_goods.

Для того, щоб зрозуміти якому клієнту належить конкретна програма лояльності, виникає зв’язок 1:N, між Loyalty_program та Client. Це зумовило створення зовнішнього ключа client_id в таблиці Loyalty_program.

Для того, щоб зрозуміти якому клієнту належить конкретне замовлення, виникає зв’язок 1:N, між Order та Client. Це зумовило створення зовнішнього ключа client_id в таблиці Order.

Для того, щоб зрозуміти які товари належать конкретному замовленню, виник зв’язок 1:N, між Order_goods та Goods. Це зумовило створення зовнішнього ключа goods_id в таблиці Order_goods.

Для того, щоб зрозуміти яким замовленням належить конкретний товар, виник зв’язок 1:N, між Order_goods та Order. Це зумовило створення зовнішнього ключа order_id в таблиці Order_goods.

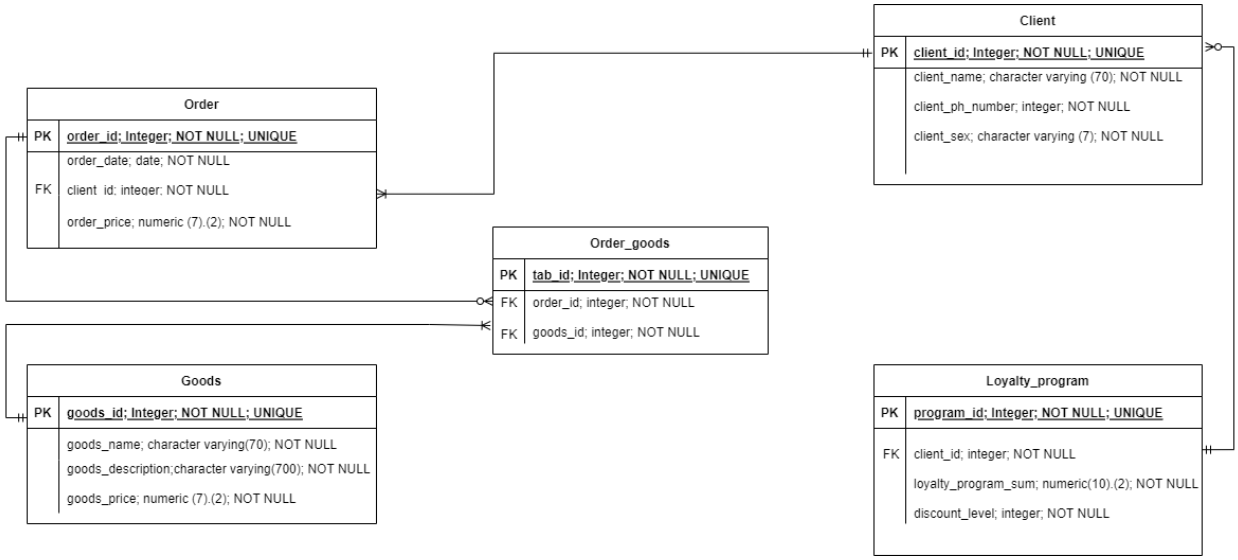

Рисунок 2 – Логічна модель даних

Середовище та компоненти розробки

Для розробки використовувалось середовище розробки PyCharm, мова програмування Python, а також бібліотека ***psycopg2.*** Даний набір інструментів для розробки забепечив коректне виконання роботи, з функціоналом, згідно варіанту.

Шаблон проектування, структура програми, її опис

У даній програмі був використаний шаблон проектування **MVC**.

Model – представляє клас, що описує логіку використовуваних даних.

View – консольний інтерфейс з яким буде взаємодіяти наш користувач.

Controller – представляє клас, що забезпечує зв'язок між консольним інтерфейсом та логікою програми.

Файли, з яких складається наша програма:

- model.py;
- view.py;
- controller.py;
- main.py.

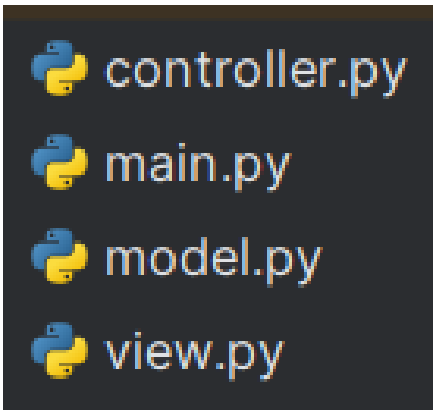

Рисунок 3 – список файлів програми

У файлі ***model.py*** описана логічна складова програми (низькорівневі запити до БД, регулювання підключень та виконання запитів, тощо).

У файлі ***view.py*** описаний консольний візуальний інтерфейс програми (приймає введені дані для запитів, виводить на екран різні повідомлення, візуальне сприйняття користувача).

У файлі ***controller.py*** описані методи, які поєднують у собі використання методів консольного інтерфейсу та логічних складових програми. (наприклад, передає введені користувачем дані в якості аргументів у методи логічної частини (класу Model) перевіряє їх на коректність та викликає їх)

У файлі ***main.py*** запускається контролер і власне вся програма починає свою роботу саме з даного файлу.

Основною функцією для нашої програми є підрахунок суми вартостей всіх замовлень та надання певного рівня знижки для кожного унікального клієнта на наступні його замовлення, в залежності від рівня знижки, який в свою чергу залежить від суми усіх замовлень. Також не мало важливим є грамотне списання або нарахування коштів та зміна рівня

знижки (з урахуванням рівня знижки) при редагуванні або видаленні замовлень.

Структура меню програми

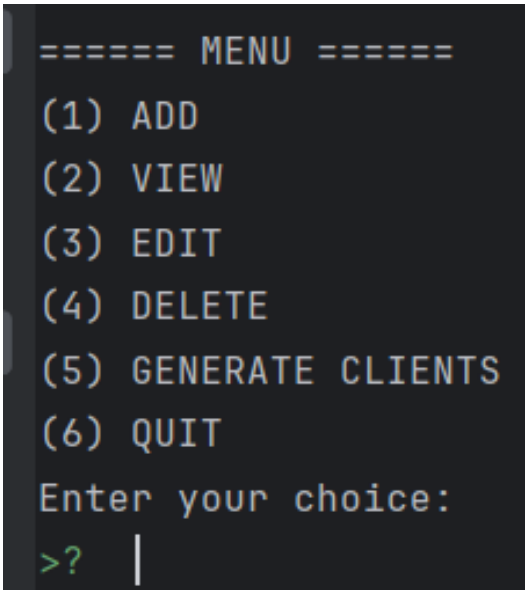

Рисунок 4 – Структура меню

При запуску програми, перед користувачем з'являється меню програми, яке містить 6 пунктів. При натисканні вказаної в дужках зліва клавіші, відкривається новий список, для виконання над пунктами цього списку дій з відповідною назвою (додавання запису, перегляд записів таблиці, редагування запису в таблиці, видалення запису з таблиці, генерація рандомних записів у таблицю, вихід з меню).

Фрагменти коду для виконання запропонованих за варіантом дій

Фрагмент коду для виконання дій над таблицею Goods (товари):

```
def add_goods(self, name, description, price):
        c = self.conn.cursor()
        c.execute('INSERT INTO "Goods" ("goods_name", "goods_description",
"goods_price") VALUES (%s, %s, %s)', (name, description, price))
        self.conn.commit()
        print("Good was added successfully!")
```

```
def edit_goods(self, goods_name, goods_description, goods_price, goods_id):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Goods" WHERE "goods_id" = %s',
(goods_id,))
        check_1 = c.fetchall()
        if check_1:
            c.execute('UPDATE "Goods" SET "goods_name"=%s,
"goods_description"=%s, "goods_price"=%s WHERE "goods_id"=%s', (goods_name,
goods_description, goods_price, goods_id))
            self.conn.commit()
            print("Good was edited successfully!")
        else:
            print("Error. This good doesn't exist.")


def delete_goods(self, goods_id):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Goods" WHERE "goods_id" = %s',
(goods_id,))
        check_1 = c.fetchall()
        c.execute('SELECT * FROM "Order_goods" WHERE "goods_id" = %s',
(goods_id,))
        check_2 = c.fetchall()
        if check_1 and check_2:
            c.execute('DELETE FROM "Order_goods" WHERE "goods_id"=%s',
(goods_id,))
            self.conn.commit()
            c.execute('DELETE FROM "Goods" WHERE "goods_id"=%s',
(goods_id,))
            self.conn.commit()
            print("Good was deleted successfully!")
        else:
            print("Error. This good doesn't exist.")


def get_all_goods(self):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Goods"')
        return c.fetchall()
```

Фрагмент коду для виконання дій над таблицею Order (замовлення):

```
def add_order(self, order_date, tmp, client_id):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Client" WHERE "client_id" = %s',
(client_id,))
        check = c.fetchall()
        if not check:
            print("Error. No such client id.")
        else:
            sum = 0
            id_array = []
            for i in range(int(tmp)):
                tmp_1 = self.get_goods_id_for_order(i)
                id_array.append(tmp_1)
```

```
                c.execute('SELECT "goods_price" FROM "Goods" WHERE
"goods_id" = %s', (tmp_1,))
                result = c.fetchone()
                sum = sum + result[0]
            c.execute('SELECT "loyalty_program_sum" FROM "Loyalty_program"
WHERE "client_id" = %s', (client_id,))
            buf_1 = c.fetchone()[0]
            sum_1 = int(buf_1) + sum
            discount_level, sum_2 = self.count_discount_level(sum_1, sum)
            final_sum = int(buf_1) + sum_2
            c.execute('UPDATE "Loyalty_program" SET
"loyalty_program_sum"=%s, "discount_level"=%s WHERE "client_id"=%s',
(final_sum, discount_level, client_id))
            self.conn.commit()
            c.execute('INSERT INTO "Order" ("order_date", "order_price",
client_id) VALUES (%s, %s, %s) RETURNING "order_id"', (order_date, sum_2,
client_id))
            buf = c.fetchone()[0]
            self.conn.commit()
            for i in range(int(tmp)):
                c.execute('INSERT INTO "Order_goods" ("goods_id",
"order_id") VALUES (%s, %s)', (id_array[i], buf))
            self.conn.commit()
            print("Order was added successfully!")



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



def edit_order(self, order_date, tmp, client_id, order_id):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Order" WHERE "order_id" = %s',
(order_id,))
        check_1 = c.fetchall()
        if check_1:
            sum = 0
            id_array = []
            for i in range(int(tmp)):
                tmp_1 = self.get_goods_id_for_order(i)
                id_array.append(tmp_1)
                c.execute('SELECT "goods_price" FROM "Goods" WHERE
"goods_id" = %s', (tmp_1,))
                result = c.fetchone()
                sum = sum + result[0]
            c.execute('SELECT "client_id" FROM "Order" WHERE "order_id" =
%s', (order_id,))
            tmp_2 = c.fetchone()[0]
            c.execute('SELECT "loyalty_program_sum" FROM "Loyalty_program"
WHERE "client_id" = %s', (tmp_2,))
            buf_1 = c.fetchone()[0]
            c.execute('SELECT "order_price" FROM "Order" WHERE "order_id" =
%s', (order_id,))
            buf_2 = c.fetchone()[0]
            sum_2 = int(buf_1) - int(buf_2)
            c.execute('UPDATE "Loyalty_program" SET
"loyalty_program_sum"=%s WHERE "client_id"=%s',(sum_2, tmp_2))
```

```
            self.conn.commit()
            c.execute('SELECT "loyalty_program_sum" FROM "Loyalty_program"
WHERE "client_id" = %s', (client_id,))
            tmp_3 = c.fetchone()[0]
            sum_3 = int(tmp_3) + int(sum)
            discount_level_1, sum_4 = self.count_discount_level(sum_3, sum)
            final_sum = int(tmp_3) + int(sum_4)
            c.execute('UPDATE "Loyalty_program" SET
"loyalty_program_sum"=%s, "discount_level"=%s WHERE
"client_id"=%s',(final_sum, discount_level_1, client_id))
            self.conn.commit()
            c.execute('UPDATE "Order" SET "order_date"=%s,
"order_price"=%s, "client_id"=%s WHERE "order_id"=%s',(order_date,
int(sum_4), client_id, order_id))
            self.conn.commit()
            c.execute('DELETE FROM "Order_goods" WHERE "order_id"=%s',
(order_id,))
            self.conn.commit()
            for i in range(int(tmp)):
                c.execute('INSERT INTO "Order_goods" ("goods_id",
"order_id") VALUES (%s, %s)', (id_array[i], order_id))
                self.conn.commit()
            print("Order was edited successfully!")
        else:
            print("Error. No such order id.")



def delete_order(self, order_id):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Order" WHERE "order_id" = %s',
(order_id,))
        check_1 = c.fetchall()
        c.execute('SELECT * FROM "Order_goods" WHERE "order_id" = %s',
(order_id,))
        check_2 = c.fetchall()
        if check_1 and check_2:
            c.execute('SELECT "client_id" FROM "Order" WHERE "order_id" =
%s', (order_id,))
            tmp_2 = c.fetchone()[0]
            c.execute('SELECT "loyalty_program_sum" FROM "Loyalty_program"
WHERE "client_id" = %s', (tmp_2,))
            buf_1 = c.fetchone()[0]
            c.execute('SELECT "order_price" FROM "Order" WHERE "order_id" =
%s', (order_id,))
            buf_2 = c.fetchone()[0]
            sum_2 = int(buf_1) - int(buf_2)
            discount_level, non_active = self.count_discount_level(sum_2,
buf_2)
            c.execute('UPDATE "Loyalty_program" SET
"loyalty_program_sum"=%s, "discount_level"=%s WHERE "client_id"=%s',
(sum_2, discount_level, tmp_2))
            self.conn.commit()
            c.execute('DELETE FROM "Order_goods" WHERE "order_id"=%s',
(order_id,))
            self.conn.commit()
            c.execute('DELETE FROM "Order" WHERE "order_id"=%s',
(order_id,))
            self.conn.commit()
            print("Order was deleted successfully!")
        else:
```

```
            print("Error. This order doesn't exist.")
```

Фрагмент коду для виконання дій над таблицею Client (клієнт):

```
def add_client(self, client_name, client_ph_number, client_sex):
        c = self.conn.cursor()
        c.execute('INSERT INTO "Client" ("client_name", "client_ph_number",
"client_sex") VALUES (%s, %s, %s) RETURNING "client_id"',(client_name,
client_ph_number, client_sex))
        result = c.fetchone()[0]
        tmp = 0
        c.execute('INSERT INTO "Loyalty_program" ("client_id",
"loyalty_program_sum", "discount_level") VALUES (%s, %s, %s)',(result, tmp,
tmp))
        self.conn.commit()
        print("Client was added successfully!")



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



def edit_client(self, client_id, client_name, client_ph_number,
client_sex):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Client" WHERE "client_id" = %s',
(client_id,))
        check = c.fetchall()
        if check:
            c.execute('UPDATE "Client" SET "client_name"=%s,
"client_ph_number"=%s, "client_sex"=%s WHERE "client_id"=%s', (client_name,
client_ph_number, client_sex, client_id))
            self.conn.commit()
            print("Client was edited successfully!")
        else:
            print("Error. This client doesn't exist.")


def delete_client(self, client_id):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Client" WHERE "client_id" = %s',
(client_id,))
        check_1 = c.fetchall()
        c.execute('SELECT * FROM "Order" WHERE "client_id" = %s',
(client_id,))
        check_2 = c.fetchall()
        c.execute('SELECT * FROM "Loyalty_program" WHERE "client_id" = %s',
(client_id,))
        check_3 = c.fetchall()
        if check_1 and check_2 and check_3:
            c.execute('DELETE FROM "Loyalty_program" WHERE "client_id"=
%s', (client_id,))
            self.conn.commit()
            c.execute('SELECT "order_id" FROM "Order" WHERE "client_id" =
%s', (client_id,))
            tmp = c.fetchall()
```

```
            for tmp_1 in tmp:
                c.execute('DELETE FROM "Order_goods" WHERE "order_id"=%s',
(tmp_1[0],))
                self.conn.commit()
            c.execute('DELETE FROM "Order" WHERE "client_id"=%s',
(client_id,))
            self.conn.commit()
            c.execute('DELETE FROM "Client" WHERE "client_id"=%s',
(client_id,))
            self.conn.commit()
            print("Client was deleted successfully!")
        elif check_1 and not check_2 and check_3:
            c.execute('DELETE FROM "Loyalty_program" WHERE "client_id"=%s',
(client_id,))
            self.conn.commit()
            c.execute('DELETE FROM "Client" WHERE "client_id"=%s',
(client_id,))
            self.conn.commit()
            print("Client was deleted successfully!")
        else:
            print("Error. This client doesn't exist.")
```

Фрагмент коду для виконання дій над таблицею Loyalty program (програма лояльності):

```
def get_all_loyalty_programs(self):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Loyalty_program"')
        return c.fetchall()
```

Фрагмент коду для автоматичного генерування записів у таблицю Client:

```
def generate_random_client(self, number):
        c = self.conn.cursor()
        for i in range(int(number)):
            c.execute('INSERT INTO "Client" ("client_name",
"client_ph_number", "client_sex") SELECT SUBSTRING(MD5(random()::text), 1,
10) AS client_name, CAST(LPAD(FLOOR(random() * 1000000000)::text, 9, \'0\')
AS integer) AS client_ph_number, CASE WHEN random() > 0.5 THEN \'male\'
ELSE \'female\' END AS client_sex RETURNING "client_id"')
            result = c.fetchone()[0]
            tmp = 0
            c.execute('INSERT INTO "Loyalty_program" ("client_id",
"loyalty_program_sum", "discount_level") VALUES (%s, %s, %s)', (result,
tmp, tmp))
            self.conn.commit()
        print("{} new random clients was generated
successfully!".format(int(number)))
```

Фрагмент коду контролера (controller.py), в якому наведено виклики функцій для таблиці Goods (товари) та значення, які їм передаються:

```
def add_goods(self):
        goods_name = self.view.get_goods_name()
```

```
        goods_description = self.view.get_goods_description()
        goods_price = self.view.get_goods_price()
        self.model.add_goods(goods_name, goods_description, goods_price)


def view_goods(self):
        goods = self.model.get_all_goods()
        self.view.show_goods(goods)


def edit_goods(self):
        goods_id_1 = self.view.get_goods_id()
        goods_name = self.view.get_goods_name()
        goods_description = self.view.get_goods_description()
        goods_price = self.view.get_goods_price()
        goods_id = goods_id_1
        if goods_price.isdigit() and goods_id.isdigit():
            self.model.edit_goods(goods_name, goods_description,
goods_price, goods_id)
        else:
            print("Error. You entered an incorrect type.")


def delete_goods(self):
        goods_id = self.view.get_goods_id()
        if goods_id.isdigit():
            self.model.delete_goods(goods_id)
        else:
            print("Error. Entered id is incorrect.")
```

Фрагмент коду контролера (controller.py), в якому наведено виклики функцій для таблиці Loyalty_program (програма лояльності) та значення, які їм передаються:

```
def add_loyalty_program(self):
        print("You can't manual add Loyalty program, because it
automaticaly generates for each new client.")


def view_loyalty_program(self):
        loyalty_program = self.model.get_all_loyalty_programs()
        self.view.show_loyalty_program(loyalty_program)


def edit_loyalty_program(self):
        print("You can't manual edit Loyalty program, because it
automaticaly generates for each new client.")


def delete_loyalty_program(self):
        print("Loyalty program can be deleted only together with the
client-owner of this program.")
```

Фрагмент коду контролера (controller.py), в якому наведено виклики функцій для таблиці Order (Замовлення) та значення, які їм передаються:

```
def add_order(self):
        order_date = self.view.get_order_date()
        client_id = self.view.get_client_id()
        tmp = self.view.get_numb_of_pos()
        if tmp.isdigit() and client_id.isdigit() and tmp.isdigit:
            self.model.add_order(order_date, tmp, client_id)
        else:
            print("Error. You entered an incorrect type of data.")


def view_order(self):
        order = self.model.get_all_order()
        self.view.show_order(order)


def edit_order(self):
        order_id_1 = self.view.get_order_id()
        order_date = self.view.get_order_date()
        client_id = self.view.get_client_id()
        tmp = self.view.get_numb_of_pos()
        order_id = order_id_1
        if order_id.isdigit() and client_id.isdigit() and tmp.isdigit():
            self.model.edit_order(order_date, tmp, client_id, order_id)
        else:
            print("Error. You entered an incorrect type of data.")


def delete_order(self):
        order_id = self.view.get_order_id()
        if order_id.isdigit():
            self.model.delete_order(order_id)
        else:
            print("Error. You entered an incorrect type of data.")
```

Фрагмент коду контролера (controller.py), в якому наведено виклики функцій для таблиці Client (Клієнт) та значення, які їм передаються:

```
def add_client(self):
        client_name= self.view.get_client_name()
        client_ph_number = self.view.get_client_ph_number()
        client_sex = self.view.get_client_sex()
        if client_name.isalpha() and client_ph_number.isdigit() and
client_sex.isalpha():
            self.model.add_client(client_name, client_ph_number,
client_sex)
        else:
            print("Error. You entered an incorrect type of data.")


def view_client(self):
        clients = self.model.get_all_clients()
        self.view.show_client(clients)
```

```
def edit_client(self):
        client_id = self.view.get_client_id()
        client_name = self.view.get_client_name()
        client_ph_number = self.view.get_client_ph_number()
        client_sex = self.view.get_client_sex()
        if client_id.isdigit() and client_name.isalpha() and
client_ph_number.isdigit() and client_sex.isalpha():
            self.model.edit_client(client_id, client_name,
client_ph_number, client_sex)
        else:
            print("Error. You entered an incorrect type of data.")


def delete_client(self):
        client_id = self.view.get_client_id()
        if client_id.isdigit():
            self.model.delete_client(client_id)
        else:
            print("Error. You entered an incorrect type of data.")
```

Фрагмент коду контролера (controller.py), в якому наведено виклик функції для автоматичної генерації записів в таблицю Client та значення, яке їй передається:

```
def run_generate(self):
        number = self.view.get_rand_number()
        if number.isdigit():
            self.model.generate_random_client(number)
        else:
            print("Error. You entered an incorrect type of data.")
```

Повний текст програми

Файл “model.py”:

```
import psycopg2
class Model:
    def __init__(self):
        self.conn = psycopg2.connect(
            dbname='Database_1',
            user='postgres',
            password='K201298k?',
            host='localhost',
            port=5432
        )

    # Goods
    def add_goods(self, name, description, price):
        c = self.conn.cursor()
        c.execute('INSERT INTO "Goods" ("goods_name", "goods_description",
"goods_price") VALUES (%s, %s, %s)', (name, description, price))
        self.conn.commit()
        print("Good was added successfully!")
```

```
    def edit_goods(self, goods_name, goods_description, goods_price,
goods_id):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Goods" WHERE "goods_id" = %s',
(goods_id,))
        check_1 = c.fetchall()
        if check_1:
            c.execute('UPDATE "Goods" SET "goods_name"=%s,
"goods_description"=%s, "goods_price"=%s WHERE "goods_id"=%s', (goods_name,
goods_description, goods_price, goods_id))
            self.conn.commit()
            print("Good was edited successfully!")
        else:
            print("Error. This good doesn't exist.")

    def delete_goods(self, goods_id):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Goods" WHERE "goods_id" = %s',
(goods_id,))
        check_1 = c.fetchall()
        c.execute('SELECT * FROM "Order_goods" WHERE "goods_id" = %s',
(goods_id,))
        check_2 = c.fetchall()
        if check_1 and check_2:
            c.execute('DELETE FROM "Order_goods" WHERE "goods_id"=%s',
(goods_id,))
            self.conn.commit()
            c.execute('DELETE FROM "Goods" WHERE "goods_id"=%s',
(goods_id,))
            self.conn.commit()
            print("Good was deleted successfully!")
        else:
            print("Error. This good doesn't exist.")

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

    # Order
    def add_order(self, order_date, tmp, client_id):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Client" WHERE "client_id" = %s',
(client_id,))
        check = c.fetchall()
        if not check:
            print("Error. No such client id.")
        else:
            sum = 0
            id_array = []
            for i in range(int(tmp)):
                tmp_1 = self.get_goods_id_for_order(i)
                id_array.append(tmp_1)
                c.execute('SELECT "goods_price" FROM "Goods" WHERE
"goods_id" = %s', (tmp_1,))
                result = c.fetchone()
                sum = sum + result[0]
            c.execute('SELECT "loyalty_program_sum" FROM "Loyalty_program"
WHERE "client_id" = %s', (client_id,))
            buf_1 = c.fetchone()[0]
            sum_1 = int(buf_1) + sum
```

```
            discount_level, sum_2 = self.count_discount_level(sum_1, sum)
            final_sum = int(buf_1) + sum_2
            c.execute('UPDATE "Loyalty_program" SET
"loyalty_program_sum"=%s, "discount_level"=%s WHERE "client_id"=%s',
(final_sum, discount_level, client_id))
            self.conn.commit()
            c.execute('INSERT INTO "Order" ("order_date", "order_price",
client_id) VALUES (%s, %s, %s) RETURNING "order_id"', (order_date, sum_2,
client_id))
            buf = c.fetchone()[0]
            self.conn.commit()
            for i in range(int(tmp)):
                c.execute('INSERT INTO "Order_goods" ("goods_id",
"order_id") VALUES (%s, %s)', (id_array[i], buf))
            self.conn.commit()
            print("Order was added successfully!")

    def get_goods_id_for_order(self, buf):
        goods_id_for_order = input("Enter id of the {} position (good):
".format(buf + 1))
        return goods_id_for_order

    def count_discount_level(self, sum_1, sum):
        if int(sum_1) < 1500:
            return 0, int(sum)*1
        elif 1500 < int(sum_1) < 5000:
            return 1, int(sum)*0.95
        elif 5000 < int(sum_1) < 10000:
            return 2, int(sum)*0.9
        elif int(sum_1) > 10000:
            return 3, int(sum)*0.85

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

    def edit_order(self, order_date, tmp, client_id, order_id):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Order" WHERE "order_id" = %s',
(order_id,))
        check_1 = c.fetchall()
        if check_1:
            sum = 0
            id_array = []
            for i in range(int(tmp)):
                tmp_1 = self.get_goods_id_for_order(i)
                id_array.append(tmp_1)
                c.execute('SELECT "goods_price" FROM "Goods" WHERE
"goods_id" = %s', (tmp_1,))
                result = c.fetchone()
                sum = sum + result[0]
            c.execute('SELECT "client_id" FROM "Order" WHERE "order_id" =
%s', (order_id,))
            tmp_2 = c.fetchone()[0]
            c.execute('SELECT "loyalty_program_sum" FROM "Loyalty_program"
WHERE "client_id" = %s', (tmp_2,))
            buf_1 = c.fetchone()[0]
            c.execute('SELECT "order_price" FROM "Order" WHERE "order_id" =
%s', (order_id,))
            buf_2 = c.fetchone()[0]
```

```
            sum_2 = int(buf_1) - int(buf_2)
            c.execute('UPDATE "Loyalty_program" SET
"loyalty_program_sum"=%s WHERE "client_id"=%s',(sum_2, tmp_2))
            self.conn.commit()
            c.execute('SELECT "loyalty_program_sum" FROM "Loyalty_program"
WHERE "client_id" = %s', (client_id,))
            tmp_3 = c.fetchone()[0]
            sum_3 = int(tmp_3) + int(sum)
            discount_level_1, sum_4 = self.count_discount_level(sum_3, sum)
            final_sum = int(tmp_3) + int(sum_4)
            c.execute('UPDATE "Loyalty_program" SET
"loyalty_program_sum"=%s, "discount_level"=%s WHERE
"client_id"=%s',(final_sum, discount_level_1, client_id))
            self.conn.commit()
            c.execute('UPDATE "Order" SET "order_date"=%s,
"order_price"=%s, "client_id"=%s WHERE "order_id"=%s',(order_date,
int(sum_4), client_id, order_id))
            self.conn.commit()
            c.execute('DELETE FROM "Order_goods" WHERE "order_id"=%s',
(order_id,))
            self.conn.commit()
            for i in range(int(tmp)):
                c.execute('INSERT INTO "Order_goods" ("goods_id",
"order_id") VALUES (%s, %s)', (id_array[i], order_id))
                self.conn.commit()
            print("Order was edited successfully!")
        else:
            print("Error. No such order id.")

    def delete_order(self, order_id):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Order" WHERE "order_id" = %s',
(order_id,))
        check_1 = c.fetchall()
        c.execute('SELECT * FROM "Order_goods" WHERE "order_id" = %s',
(order_id,))
        check_2 = c.fetchall()
        if check_1 and check_2:
            c.execute('SELECT "client_id" FROM "Order" WHERE "order_id" =
%s', (order_id,))
            tmp_2 = c.fetchone()[0]
            c.execute('SELECT "loyalty_program_sum" FROM "Loyalty_program"
WHERE "client_id" = %s', (tmp_2,))
            buf_1 = c.fetchone()[0]
            c.execute('SELECT "order_price" FROM "Order" WHERE "order_id" =
%s', (order_id,))
            buf_2 = c.fetchone()[0]
            sum_2 = int(buf_1) - int(buf_2)
            discount_level, non_active = self.count_discount_level(sum_2,
buf_2)
            c.execute('UPDATE "Loyalty_program" SET
"loyalty_program_sum"=%s, "discount_level"=%s WHERE "client_id"=%s',
(sum_2, discount_level, tmp_2))
            self.conn.commit()
            c.execute('DELETE FROM "Order_goods" WHERE "order_id"=%s',
(order_id,))
            self.conn.commit()
            c.execute('DELETE FROM "Order" WHERE "order_id"=%s',
(order_id,))
            self.conn.commit()
            print("Order was deleted successfully!")
```

```
        else:
            print("Error. This order doesn't exist.")


    # Client

    def add_client(self, client_name, client_ph_number, client_sex):
        c = self.conn.cursor()
        c.execute('INSERT INTO "Client" ("client_name", "client_ph_number",
"client_sex") VALUES (%s, %s, %s) RETURNING "client_id"',(client_name,
client_ph_number, client_sex))
        result = c.fetchone()[0]
        tmp = 0
        c.execute('INSERT INTO "Loyalty_program" ("client_id",
"loyalty_program_sum", "discount_level") VALUES (%s, %s, %s)',(result, tmp,
tmp))
        self.conn.commit()
        print("Client was added successfully!")

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

    def edit_client(self, client_id, client_name, client_ph_number,
client_sex):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Client" WHERE "client_id" = %s',
(client_id,))
        check = c.fetchall()
        if check:
            c.execute('UPDATE "Client" SET "client_name"=%s,
"client_ph_number"=%s, "client_sex"=%s WHERE "client_id"=%s', (client_name,
client_ph_number, client_sex, client_id))
            self.conn.commit()
            print("Client was edited successfully!")
        else:
            print("Error. This client doesn't exist.")

    def delete_client(self, client_id):
        c = self.conn.cursor()
        c.execute('SELECT * FROM "Client" WHERE "client_id" = %s',
(client_id,))
        check_1 = c.fetchall()
        c.execute('SELECT * FROM "Order" WHERE "client_id" = %s',
(client_id,))
        check_2 = c.fetchall()
        c.execute('SELECT * FROM "Loyalty_program" WHERE "client_id" = %s',
(client_id,))
        check_3 = c.fetchall()
        if check_1 and check_2 and check_3:
            c.execute('DELETE FROM "Loyalty_program" WHERE "client_id"=
%s', (client_id,))
            self.conn.commit()
            c.execute('SELECT "order_id" FROM "Order" WHERE "client_id" =
%s', (client_id,))
            tmp = c.fetchall()
            for tmp_1 in tmp:
                c.execute('DELETE FROM "Order_goods" WHERE "order_id"=%s',
(tmp_1[0],))
                self.conn.commit()
```

```
            c.execute('DELETE FROM "Order" WHERE "client_id"=%s',
(client_id,))
            self.conn.commit()
            c.execute('DELETE FROM "Client" WHERE "client_id"=%s',
(client_id,))
            self.conn.commit()
            print("Client was deleted successfully!")
        elif check_1 and not check_2 and check_3:
            c.execute('DELETE FROM "Loyalty_program" WHERE "client_id"=%s',
(client_id,))
            self.conn.commit()
            c.execute('DELETE FROM "Client" WHERE "client_id"=%s',
(client_id,))
            self.conn.commit()
            print("Client was deleted successfully!")
        else:
            print("Error. This client doesn't exist.")

    # Loyalty_program

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

    # Auto random generating clients

    def generate_random_client(self, number):
        c = self.conn.cursor()
        for i in range(int(number)):
            c.execute('INSERT INTO "Client" ("client_name",
"client_ph_number", "client_sex") SELECT SUBSTRING(MD5(random()::text), 1,
10) AS client_name, CAST(LPAD(FLOOR(random() * 1000000000)::text, 9, \'0\')
AS integer) AS client_ph_number, CASE WHEN random() > 0.5 THEN \'male\'
ELSE \'female\' END AS client_sex RETURNING "client_id"')
            result = c.fetchone()[0]
            tmp = 0
            c.execute('INSERT INTO "Loyalty_program" ("client_id",
"loyalty_program_sum", "discount_level") VALUES (%s, %s, %s)', (result,
tmp, tmp))
            self.conn.commit()
        print("{} new random clients was generated
successfully!".format(int(number)))
```

Файл “view.py”:

```
class View:

    # Goods
    def show_goods(self, Goods):
        if Goods:
            print("Goods:")
            for goods in Goods:
                print(f"goods_id: {goods[0]}, goods_name: {goods[1]},
goods_description: {goods[2]}, goods_price: {goods[3]}")
        else:
            print("Good was not found.")
```

```
    def get_goods_name(self):
        goods_name = input("Enter goods name: ")
        return goods_name

    def get_goods_id(self):
        goods_id = input("Enter goods id: ")
        return goods_id

    def get_goods_description(self):
        goods_description = input("Enter goods description: ")
        return goods_description

    def get_goods_price(self):
        goods_price = input("Enter goods price: ")
        return goods_price

    # Order
    def show_order(self, Order):
        if Order:
            print("Orders:")
            for order in Order:
                print(f"order_id: {order[0]}, order_date: {order[1]},
order_price: {order[2]}, client_id: {order[3]}")
        else:
            print("Order was not found.")

    def get_order_date(self):
        order_date = input("Enter order date: ")
        return order_date

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

    def get_order_price(self):
        order_price = input("Enter order price: ")
        return order_price

    def get_numb_of_pos(self):
        pos_numb = input("Enter number of positions in this order: ")
        return pos_numb

    # Client
    def show_client(self, Client):
        if Client:
            print("Clients:")
            for client in Client:
                print(f"Client id: {client[0]}, client name: {client[1]},
client phone number: {client[2]}, client sex: {client[3]}")
        else:
            print("Client was not found.")

    def get_client_id(self):
        client_id = input("Enter client id: ")
        return client_id

    def get_client_name(self):
        client_name = input("Enter client name: ")
        return client_name

    def get_client_ph_number(self):
```

```
        client_ph_number = input("Enter client phone number: ")
        return client_ph_number

    def get_client_sex(self):
        client_sex = input("Enter client sex: ")
        return client_sex

    # Loyalty program
    def show_loyalty_program(self, Loyalty_program):
        if Loyalty_program:
            print("Loyalty programs:")
            for program in Loyalty_program:
                print(f"Loyalty program id: {program[0]}, client_id:
{program[1]}, loyalty_program_sum: {program[2]}, discount level:
{program[3]}")
        else:
            print("Loyalty program was not found.")

    def get_rand_number(self):
        rand_number = input("Enter number of random generated clients: ")
        return rand_number

    def show_message(self, message):
        print(message)
```

Файл “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:
            choice = self.show_menu()
            if choice == '1':
                self.run_add()
            elif choice == '2':
                self.run_view()
            elif choice == '3':
                self.run_edit()
            elif choice == '4':
                self.run_delete()
            elif choice == '5':
                self.run_generate()
            elif choice == '6':
                break

    def show_menu(self):
        self.view.show_message("\n====== MENU ======")
        self.view.show_message("(1) ADD")
        self.view.show_message("(2) VIEW")
        self.view.show_message("(3) EDIT")
        self.view.show_message("(4) DELETE")
        self.view.show_message("(5) GENERATE CLIENTS")
        self.view.show_message("(6) QUIT")
```

```
        return input("Enter your choice: ")

    def menu(self):
        self.view.show_message("(1) GOODS")
        self.view.show_message("(2) LOYALTY PROGRAM")
        self.view.show_message("(3) CLIENT")
        self.view.show_message("(4) ORDER")
        self.view.show_message("(5) QUIT")
        return input("Enter your choice: ")

    def run_add(self):
        self.view.show_message("\n===== ADD =====")
        while True:
            choice_1 = self.menu()
            if choice_1 == '1':
                self.add_goods()
            elif choice_1 == '2':
                self.add_loyalty_program()
            elif choice_1 == '3':
                self.add_client()
            elif choice_1 == '4':
                self.add_order()
            elif choice_1 == '5':
                break

    def run_view(self):
        self.view.show_message("\n===== VIEW =====")
        while True:
            choice_2 = self.menu()
            if choice_2 == '1':
                self.view_goods()
            elif choice_2 == '2':
                self.view_loyalty_program()
            elif choice_2 == '3':
                self.view_client()
            elif choice_2 == '4':
                self.view_order()
            elif choice_2 == '5':
                break

    def run_edit(self):
        self.view.show_message("\n===== EDIT =====")
        while True:
            choice_3 = self.menu()
            if choice_3 == '1':
                self.edit_goods()
            elif choice_3 == '2':
                self.edit_loyalty_program()
            elif choice_3 == '3':
                self.edit_client()
            elif choice_3 == '4':
                self.edit_order()
            elif choice_3 == '5':
                break

    def run_delete(self):
        self.view.show_message("\n===== DELETE =====")
        while True:
            choice_4 = self.menu()
            if choice_4 == '1':
                self.delete_goods()
```

```
            elif choice_4 == '2':
                self.delete_loyalty_program()
            elif choice_4 == '3':
                self.delete_client()
            elif choice_4 == '4':
                self.delete_order()
            elif choice_4 == '5':
                break

    def run_generate(self):
        number = self.view.get_rand_number()
        if number.isdigit():
            self.model.generate_random_client(number)
        else:
            print("Error. You entered an incorrect type of data.")

    # Goods
    def add_goods(self):
        goods_name = self.view.get_goods_name()
        goods_description = self.view.get_goods_description()
        goods_price = self.view.get_goods_price()
        self.model.add_goods(goods_name, goods_description, goods_price)

    def view_goods(self):
        goods = self.model.get_all_goods()
        self.view.show_goods(goods)

    def edit_goods(self):
        goods_id_1 = self.view.get_goods_id()
        goods_name = self.view.get_goods_name()
        goods_description = self.view.get_goods_description()
        goods_price = self.view.get_goods_price()
        goods_id = goods_id_1
        if goods_price.isdigit() and goods_id.isdigit():
            self.model.edit_goods(goods_name, goods_description,
goods_price, goods_id)
        else:
            print("Error. You entered an incorrect type.")

    def delete_goods(self):
        goods_id = self.view.get_goods_id()
        if goods_id.isdigit():
            self.model.delete_goods(goods_id)
        else:
            print("Error. Entered id is incorrect.")

    # Loyalty_program
    def add_loyalty_program(self):
        print("You can't manual add Loyalty program, because it
automaticaly generates for each new client.")

    def view_loyalty_program(self):
        loyalty_program = self.model.get_all_loyalty_programs()
        self.view.show_loyalty_program(loyalty_program)

    def edit_loyalty_program(self):
        print("You can't manual edit Loyalty program, because it
automaticaly generates for each new client.")

    def delete_loyalty_program(self):
```

```
            print("Loyalty program can be deleted only together with the
client-owner of this program.")

    # Order
    def add_order(self):
        order_date = self.view.get_order_date()
        client_id = self.view.get_client_id()
        tmp = self.view.get_numb_of_pos()
        if tmp.isdigit() and client_id.isdigit() and tmp.isdigit:
            self.model.add_order(order_date, tmp, client_id)
        else:
            print("Error. You entered an incorrect type of data.")

    def view_order(self):
        order = self.model.get_all_order()
        self.view.show_order(order)

    def edit_order(self):
        order_id_1 = self.view.get_order_id()
        order_date = self.view.get_order_date()
        client_id = self.view.get_client_id()
        tmp = self.view.get_numb_of_pos()
        order_id = order_id_1
        if order_id.isdigit() and client_id.isdigit() and tmp.isdigit():
            self.model.edit_order(order_date, tmp, client_id, order_id)
        else:
            print("Error. You entered an incorrect type of data.")

    def delete_order(self):
        order_id = self.view.get_order_id()
        if order_id.isdigit():
            self.model.delete_order(order_id)
        else:
            print("Error. You entered an incorrect type of data.")

    # Client
    def add_client(self):
        client_name= self.view.get_client_name()
        client_ph_number = self.view.get_client_ph_number()
        client_sex = self.view.get_client_sex()
        if client_name.isalpha() and client_ph_number.isdigit() and
client_sex.isalpha():
            self.model.add_client(client_name, client_ph_number,
client_sex)
        else:
            print("Error. You entered an incorrect type of data.")

    def view_client(self):
        clients = self.model.get_all_clients()
        self.view.show_client(clients)

    def edit_client(self):
        client_id = self.view.get_client_id()
        client_name = self.view.get_client_name()
        client_ph_number = self.view.get_client_ph_number()
        client_sex = self.view.get_client_sex()
        if client_id.isdigit() and client_name.isalpha() and
client_ph_number.isdigit() and client_sex.isalpha():
            self.model.edit_client(client_id, client_name,
client_ph_number, client_sex)
        else:
```

```
            print("Error. You entered an incorrect type of data.")

    def delete_client(self):
        client_id = self.view.get_client_id()
        if client_id.isdigit():
            self.model.delete_client(client_id)
        else:
            print("Error. You entered an incorrect type of data.")
```

Файл “main.py”:

```
from controller import Controller

if __name__ == "__main__":
    controller = Controller()
    controller.run()
```

Перевірка коректності роботи програми

Перевірка результатів виконання операції вставки запису в дочірню таблицю:

	order_id [PK] integer	order_date date	order_price numeric (10,2)	client_id integer
15	32	2013-12-12	600.00	14
16	33	2014-12-12	500.00	14
17	34	2023-12-03	500.00	14
18	35	2023-09-03	500.00	14
19	36	2023-02-03	500.00	14
20	37	2023-02-13	500.00	14
21	38	2003-12-12	450.00	14
22	39	2017-11-17	1235.00	9
23	40	2023-07-11	1235.00	10

	program_id [PK] integer	client_id integer	loyalty_program_sum numeric (10,2)	discount_level integer
1	2	9	2435.00	1
2	3	10	2935.00	1
3	7	14	7106.00	2
4	8	16	0.00	0
5	9	17	0.00	0
6	10	18	0.00	0
7	11	19	0.00	0
8	12	20	0.00	0
9	13	21	0.00	0

	goods_id integer	order_id integer	tab_id [PK] integer
30	3	36	58
31	3	37	59
32	3	38	60
33	1	39	63
34	2	39	64
35	3	39	65
36	1	40	68
37	2	40	69
38	3	40	70

```
===== ADD =====
(1) GOODS
(2) LOYALTY PROGRAM
(3) CLIENT
(4) ORDER
(5) QUIT
Enter your choice: >? 4
Enter order date: >? 11.12.2022
Enter client id: >? 14
Enter number of positions in this order: >? 2
Enter id of the 1 position (good): >? 1
Enter id of the 2 position (good): >? 2
Order was added successfully!
```

	order_id [PK] integer	order_date date	order_price numeric (10,2)	client_id integer
16	33	2014-12-12	500.00	14
17	34	2023-12-03	500.00	14
18	35	2023-09-03	500.00	14
19	36	2023-02-03	500.00	14
20	37	2023-02-13	500.00	14
21	38	2003-12-12	450.00	14
22	39	2017-11-17	1235.00	9
23	40	2023-07-11	1235.00	10
24	42	2022-12-11	720.00	14

	program_id [PK] integer	client_id integer	loyalty_program_sum numeric (10,2)	discount_level integer
1	2	9	2435.00	1
2	3	10	2935.00	1
3	7	14	7826.00	2
4	8	16	0.00	0
5	9	17	0.00	0
6	10	18	0.00	0
7	11	19	0.00	0
8	12	20	0.00	0
9	13	21	0.00	0

	goods_id integer	order_id integer	tab_id [PK] integer
32	3	38	60
33	1	39	63
34	2	39	64
35	3	39	65
36	1	40	68
37	2	40	69
38	3	40	70
39	1	42	76
40	2	42	77

Перевірка результатів виконання операції редагування запису:

```
===== EDIT =====
(1) GOODS
(2) LOYALTY PROGRAM
(3) CLIENT
(4) ORDER
(5) QUIT
Enter your choice: >? 4
Enter order id: >? 42
Enter order date: >? 07.07.2004
Enter client id: >? 9
Enter number of positions in this order: >? 3
Enter id of the 1 position (good): >? 1
Enter id of the 2 position (good): >? 2
Enter id of the 3 position (good): >? 3
Order was edited successfully!
```

	order_id [PK] integer	order_date date	order_price numeric (10,2)	client_id integer
16	33	2014-12-12	500.00	14
17	34	2023-12-03	500.00	14
18	35	2023-09-03	500.00	14
19	36	2023-02-03	500.00	14
20	37	2023-02-13	500.00	14
21	38	2003-12-12	450.00	14
22	39	2017-11-17	1235.00	9
23	40	2023-07-11	1235.00	10
24	42	2004-07-07	1235.00	9

	program_id [PK] integer	client_id integer	loyalty_program_sum numeric (10,2)	discount_level integer
1	2	9	3670.00	1
2	3	10	2935.00	1
3	7	14	7106.00	2
4	8	16	0.00	0
5	9	17	0.00	0
6	10	18	0.00	0
7	11	19	0.00	0
8	12	20	0.00	0
9	13	21	0.00	0

	goods_id integer	order_id integer	tab_id [PK] integer
33	1	39	63
34	2	39	64
35	3	39	65
36	1	40	68
37	2	40	69
38	3	40	70
39	1	42	78
40	2	42	79
41	3	42	80

Перевірка результатів виконання операції видалення запису:

```
===== DELETE =====
(1) GOODS
(2) LOYALTY PROGRAM
(3) CLIENT
(4) ORDER
(5) QUIT
Enter your choice: >? 4
Enter order id: >? 42
Order was deleted successfully!
```

	order_id [PK] integer	order_date date	order_price numeric (10,2)	client_id integer
15	32	2013-12-12	600.00	14
16	33	2014-12-12	500.00	14
17	34	2023-12-03	500.00	14
18	35	2023-09-03	500.00	14
19	36	2023-02-03	500.00	14
20	37	2023-02-13	500.00	14
21	38	2003-12-12	450.00	14
22	39	2017-11-17	1235.00	9
23	40	2023-07-11	1235.00	10

	goods_id integer	order_id integer	tab_id [PK] integer
30	3	36	58
31	3	37	59
32	3	38	60
33	1	39	63
34	2	39	64
35	3	39	65
36	1	40	68
37	2	40	69
38	3	40	70

	program_id [PK] integer	client_id integer	loyalty_program_sum numeric (10,2)	discount_level integer
1	2	9	2435.00	1
2	3	10	2935.00	1
3	7	14	7106.00	2
4	8	16	0.00	0
5	9	17	0.00	0
6	10	18	0.00	0
7	11	19	0.00	0
8	12	20	0.00	0
9	13	21	0.00	0

Перевірка результатів виконання операції генерування записів:

	client_id [PK] integer	client_name character varying (70)	client_ph_number integer	client_sex character varying (7)
2	10	Kateryna	987654321	female
3	11	Nazar	333333333	male
4	14	Viktor	986754323	male
5	16	ccbf20bd29	515770746	female
6	17	24a0904e75	886278482	male
7	18	7862eded10	44707546	male
8	19	5b93da87f8	508541125	female
9	20	2322e63923	902198436	female
10	21	948e4fcd77	215605797	female

```
====== MENU ======
(1) ADD
(2) VIEW
(3) EDIT
(4) DELETE
(5) GENERATE CLIENTS
(6) QUIT
Enter your choice: >? 5
Enter number of random generated clients: >? 5
5 new random clients was generated successfully!
```

	client_id [PK] integer	client_name character varying (70)	client_ph_number integer	client_sex character varying (7)
7	18	7862eded10	44707546	male
8	19	5b93da87f8	508541125	female
9	20	2322e63923	902198436	female
10	21	948e4fcd77	215605797	female
11	22	6105448e02	77415573	male
12	23	21d38c609a	710203875	female
13	24	1082a1d23e	695768085	female
14	25	778fc1226a	753346515	female
15	26	2ece67fa7a	882832058	male