НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені 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
групи КВ-13 Шандиба А. А.
Telegram: https://t.me/andriic0
Github: https://github.com/andreas778/bd

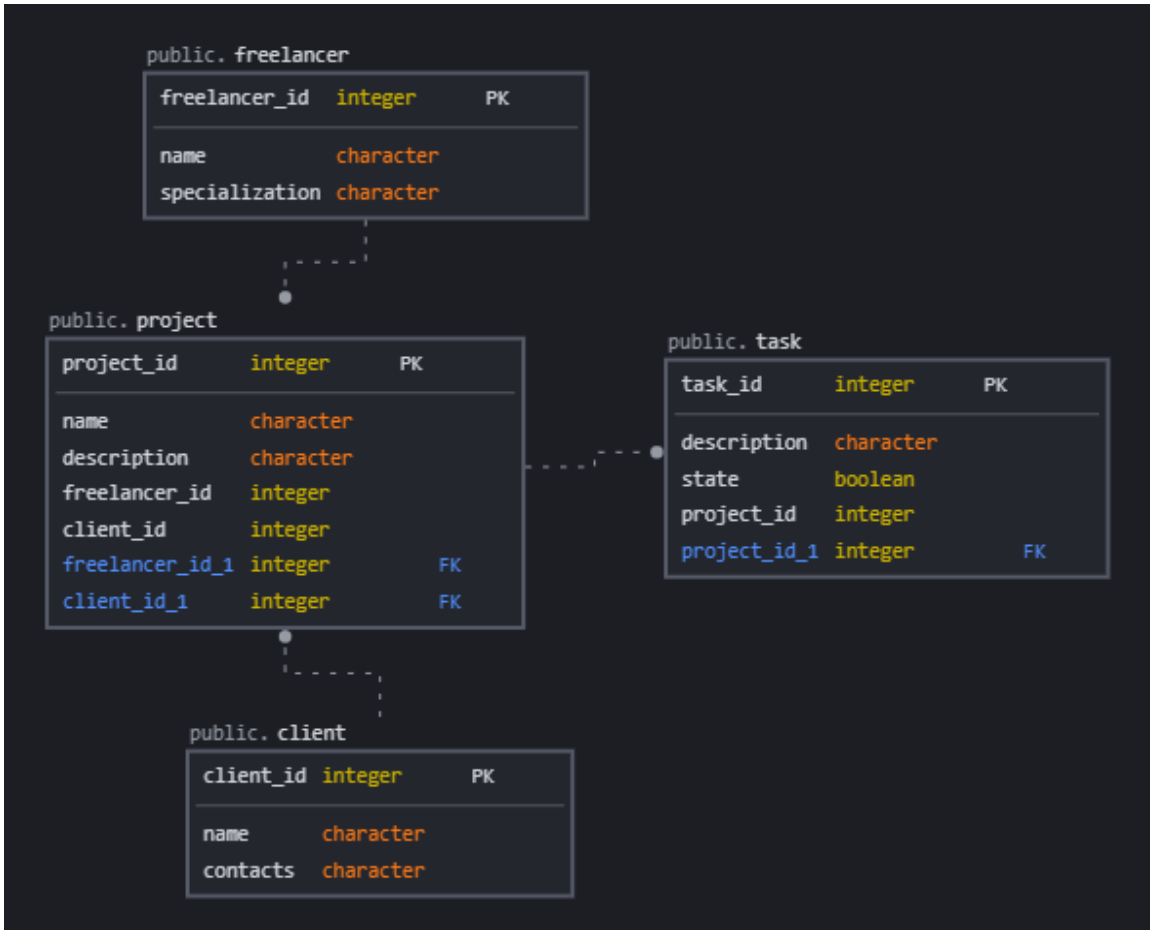
Київ – 2023

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

Загальне завдання роботи полягає у наступному:

1. Реалізувати функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

Логічна модель предметної області «Система управління завданнями та проектами для фрілансерів»



Середовище та компоненти розробки

Для виконання лабораторної роботи використовувалась мова програмування Python та середовище розробки Visual Studio. Для підключення до серверу бази даних PostgreSQL використано сторонню бібліотеку psycopg2.

Шаблон проектування

MVC – шаблон проектування, який використаний у програмі.

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

View – консольний інтерфейс, з яким буде взаємодіяти користувач. Містить компоненти, згідно яким користувач бачить необхідні дані, що є представленням даних у вигляді консольного інтерфейсу.

Controller – представляє клас, що забезпечує зв'язок між користувачем і системою, поданням і сховищем даних. Отримує введені користувачем дані і обробляє їх, в залежності від результатів обробки відправляє користувачеві певний висновок.

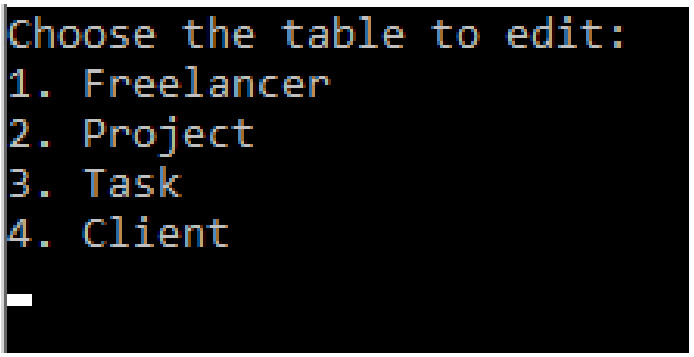
Структура та опис програми

Програма має 6 модулів:

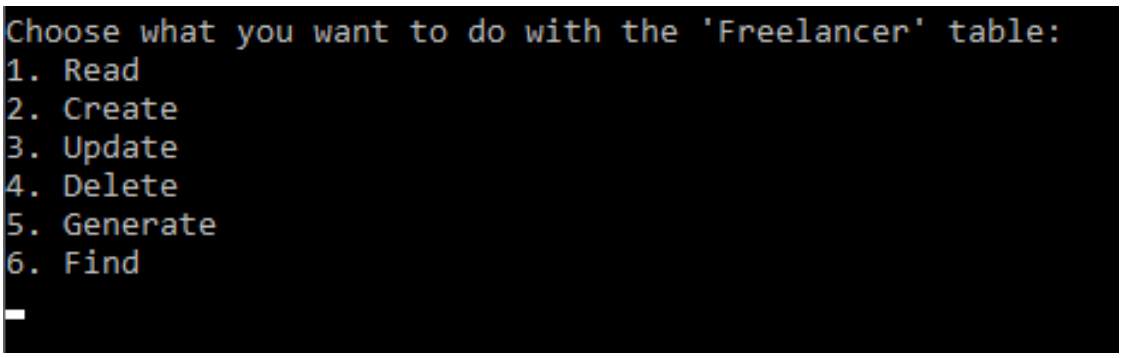
1. Main.py - точка входу до програми, що викликає методи із контролерів.
2. Controller.py - містить функції та засоби для підключення бази даних до програми.
3. Freelancer.py - контролер для таблиці Freelancer.
4. Project.py - контролер для таблиці Project.
5. Task.py - контролер для таблиці Task.
6. Client.py - контролер для таблиці Client.

Меню програми

Головне меню програми, у якому користувач обирає компонент БД, із яким надалі буде працювати, має вигляд:

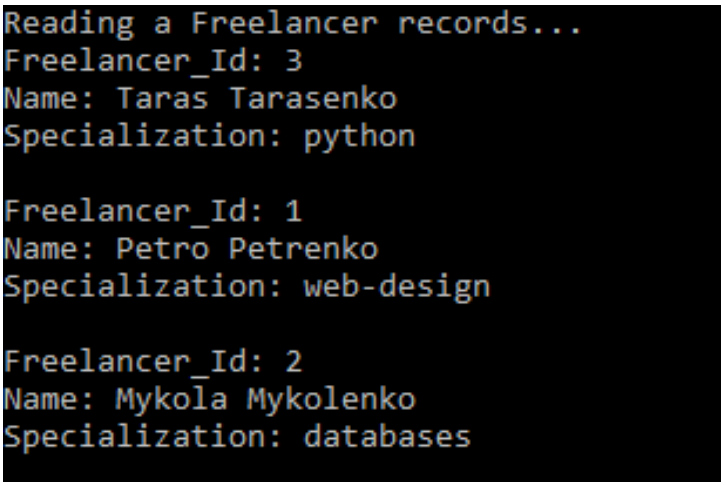


Меню, у якому користувач обирає дії, яку він хоче зробити із обраною категорією має вигляд:



Розглянемо функціональні можливості програми:

1. Читання даних:



SQL-запит:

sql_select = "SELECT Freelancer_Id, Name, Specialization FROM Freelancer"

2. Додавання даних:

```
Creating a new Freelancer record...
Freelancer_Id: 5
Name: Stepan Stepanenko
Specialization: designer
```

Результат операції додавання:

```
Reading a Freelancer records...
Freelancer_Id: 3
Name: Taras Tarasenko
Specialization: python

Freelancer_Id: 1
Name: Petro Petrenko
Specialization: web-design

Freelancer_Id: 2
Name: Mykola Mykolenko
Specialization: databases

Freelancer_Id: 5
Name: Stepan Stepanenko
Specialization: designer
```

SQL-запит:

sql_insert = "INSERT INTO Freelancer (Freelancer_Id, Name, Specialization) VALUES (%s, %s, %s)"

3. Редагування даних:

```
Updating Freelancer records...
Enter the name of the option you want to find with: Freelancer_Id
Enter the value to find: 5
Freelancer_Id: 5
Name: Stepan Stepanenko
Specialization: designer

Enter the name of the option you want to change: Specialization
Enter the new value: developer
Records updated successfully
```

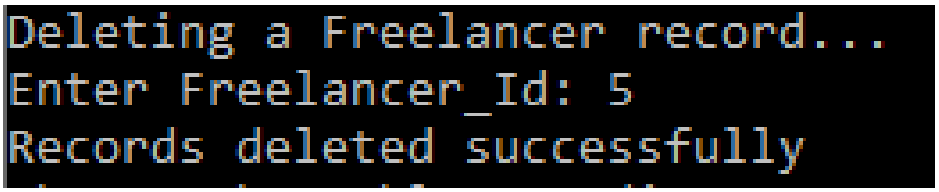
Результат операції редагування:

```
Reading a Freelancer records...
Freelancer_Id: 3
Name: Taras Tarasenko
Specialization: python

Freelancer_Id: 1
Name: Petro Petrenko
Specialization: web-design

Freelancer_Id: 2
Name: Mykola Mykolenko
Specialization: databases

Freelancer_Id: 5
Name: Stepan Stepanenko
Specialization: developer
```

SQL-запит:

sql_update = "UPDATE Freelancer SET {field_to_update} = %s WHERE {field_to_find} = %s"

4. Видалення даних:



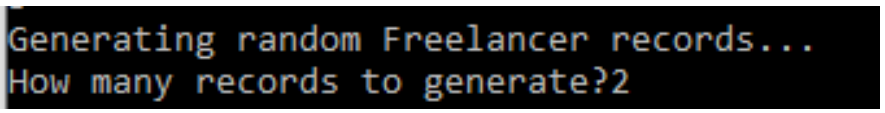
Результат операції видалення:

```
Reading a Freelancer records...
Freelancer_Id: 3
Name: Taras Tarasenko
Specialization: python

Freelancer_Id: 1
Name: Petro Petrenko
Specialization: web-design

Freelancer_Id: 2
Name: Mykola Mykolenko
Specialization: databases
```

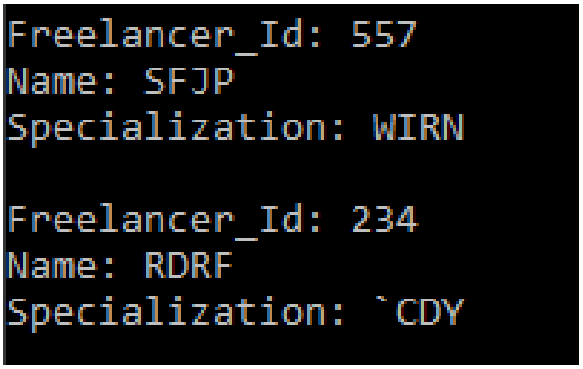
SQL-запит:

sql_delete = "DELETE FROM Freelancer WHERE Freelancer_Id = %s"

5. Генерування даних:



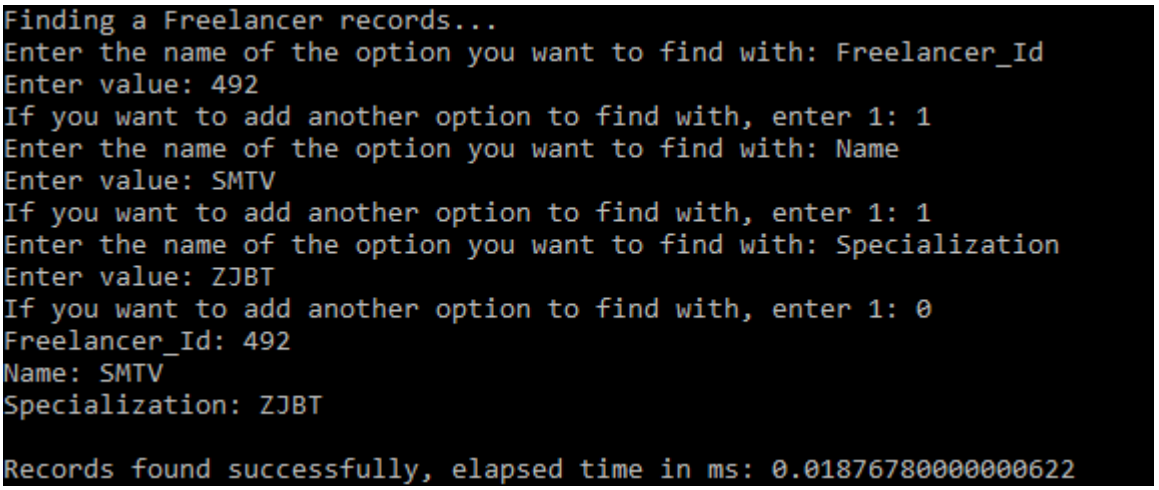
Результат операції генерування:



SQL-запит:

sql_generate = "INSERT INTO Freelancer (Freelancer_Id, Name, Specialization) (select "trunc(random()*1000)::int", "chr(trunc(65 + random()* 50)::int) || chr(trunc(65 + random() * 25)::int) || chr(trunc(65 +random() * 25)::int) || chr(trunc(65 + random() * 25)::int)", "chr(trunc(65 + random()* 50)::int) || chr(trunc(65 + random() * 25)::int) || chr(trunc(65 +random() * 25)::int) || chr(trunc(65 + random() * 25)::int)")"

6. Пошук даних за декількома атрибутами одночасно:



SQL-запит:

sql_select = "SELECT Freelancer_Id, Name, Specialization FROM Freelancer WHERE Freelancer_Id=’2’, Name=’Mykola Mykolenko’ AND Specialization=’databases’"

Код програми

Main.py

```
from Freelancer import Freelancer
from Project import Project
from Task import Task
from Client import Client



def main():
        connection_string = "dbname=postgres user=postgres password=4455 host=localhost"
    table = 0
    action = 0

    while True:
        table = first_menu()
        if table == 0:
            return
        controller = None

        if table == 1:
            name = "Freelancer"
            action = second_menu(name)
            controller = Freelancer(connection_string, name)
        elif table == 2:
            name = "Project"
            action = second_menu(name)
            controller = Project(connection_string, name)
        elif table == 3:
```

```
            name = "Task"
            action = second_menu(name)
            controller = Task(connection_string, name)
        elif table == 4:
            name = "Client"
            action = second_menu(name)
            controller = Client(connection_string, name)

        if action == 1:
            controller.read()
        elif action == 2:
            controller.create()
        elif action == 3:
            controller.update()
        elif action == 4:
            controller.delete()
        elif action == 5:
            controller.generate()
        elif action == 6:
            controller.find()

def first_menu():
    while True:
        print("Choose the table to edit:")
        print("1. Freelancer")
        print("2. Project")
        print("3. Task")
        print("4. Client")
        choice = input()
```

```
        try:
            choice = int(choice)
            if 0 < choice < 5:
                return choice
        except ValueError:
            pass


def second_menu(table_to_change):
    while True:
        print(f"Choose what you want to do with the '{table_to_change}' table:")
        print("1. Read")
        print("2. Create")
        print("3. Update")
        print("4. Delete")
        print("5. Generate")
        print("6. Find")
        choice = input()
        try:
            choice = int(choice)
            if 0 < choice < 7:
                return choice
        except ValueError:
            pass


if __name__ == "__main__":
    main()
```

Controller.py

```
import psycopg2

class Controller:
    sqlRandomString = "chr(trunc(65 + random()* 50)::int) || chr(trunc(65 + random() * 25)::int) || chr(trunc(65 +random() * 25)::int) || chr(trunc(65 + random() * 25)::int)"
    sqlRandomInteger = "trunc(random()*1000)::int"

    def __init__(self, connection_string, name):
        self.connection_string = connection_string
        self.conn = psycopg2.connect(connection_string)
        self.name = name

    def read(self, where_condition=""):
        pass

    def create(self):
        pass

    def update(self):
        print(f"Updating {self.name} records...")
        while True:
            try:
                field_to_find = input("Enter the name of the option you want to find with: ")
                value_to_find = input("Enter the value to find: ")
                self.read(f"WHERE {field_to_find} = '{value_to_find}'")
                break
```

```
            except Exception as e:
                print(e)

        while True:
            try:
                field_to_update = input("Enter the name of the option you want to change: ")
                value_to_set = input("Enter the new value: ")
                sql_update = f"UPDATE {self.name} SET {field_to_update} = %s WHERE {field_to_find} = %s"
                with self.conn, self.conn.cursor() as cur:
                    cur.execute(sql_update, (value_to_set, value_to_find))
                break
            except Exception as e:
                print(e)
        print("Records updated successfully")

    def delete(self):
        print(f"Deleting a {self.name} record...")
        while True:
            try:
                id = int(input(f"Enter {self.name}_Id: "))
                break
            except:
                print("{self.name}_Id must be Integer")
            sql_delete = f"DELETE FROM {self.name} WHERE {self.name}_Id = %s"
        with self.conn, self.conn.cursor() as cur:
            cur.execute(sql_delete, (id,))
```

```
        print("Records deleted successfully")

    def generate(self):
        pass

    def find(self):
        print(f"Finding a {self.name} records...")
        sql_find = f"WHERE "
        x = 0
        while True:
            try:
                while True:
                    option = input("Enter the name of the option you want to find with:
")
                    value = input("Enter value: ")
                    x = input("If you want to add another option to find with, enter 1: ")
                    if x == '1':
                        sql_find += f"{option} ='{value}' AND "
                    else:
                        sql_find += f"{option} ='{value}'"
                        break
                t1 = perf_counter()
                self.read(sql_find)
                t2 = perf_counter()
                print("Records found successfully, elapsed time in ms:", t2-t1)
                self.read(sql_find)
                break
            except Exception as e:
                print(e)
```

Freelancer.py

```
from Controller import Controller

class Freelancer(Controller):
    def read(self, where_condition=""):
        if not(where_condition): print(f"Reading a {self.name} records...")
            sql_select = f"SELECT Freelancer_Id, Name, Specialization FROM
{self.name} {where_condition}"
        with self.conn, self.conn.cursor() as cur:
            cur.execute(sql_select)
            for row in cur.fetchall():
                print(f"Freelancer_Id: {row[0]}")
                print(f"Name: {row[1]}")
                print(f"Specialization: {row[2]}")
                print()

    def create(self):
        print(f"Creating a new {self.name} record...")
        while True:
            try:
                freelancer_id = int(input(f"{self.name}_Id: "))
                break
            except:
                print(f"{self.name}_id must be Integer")
        name = input("Name: ")
```

```
        specialization = input("Specialization: ")
            sql_insert = f"INSERT INTO {self.name} (Freelancer_Id, Name, Specialization) VALUES (%s, %s, %s)"
        with self.conn, self.conn.cursor() as cur:
            cur.execute(sql_insert, (freelancer_id, name, specialization))


    def generate(self):
        print(f"Generating random {self.name} records...")
        while True:
            try:
                records_amount = int(input("How many records to generate?"))
                break
            except:
                print("Records amount must be Integer")
            sql_generate = f"INSERT INTO {self.name} (Freelancer_Id, Name, Specialization) (select {self.sqlRandomInteger}, {self.sqlRandomString}, {self.sqlRandomString})"
        for _ in range(records_amount):
            with self.conn, self.conn.cursor() as cur:
                cur.execute(sql_generate)
```

Project.py

```
from Controller import Controller


class Project(Controller):
    def read(self, where_condition=""):
        if not(where_condition): print(f"Reading a {self.name} records...")
            sql_select = f"SELECT Project_Id, Name, Description, Freelancer_Id,
Client_Id FROM {self.name} {where_condition}"
        with self.conn, self.conn.cursor() as cur:
            cur.execute(sql_select)
            for row in cur.fetchall():
                print(f"Project_Id: {row[0]}")
                print(f"Name: {row[1]}")
                print(f"Description: {row[2]}")
                print(f"Freelancer_Id: {row[3]}")
                print(f"Client_Id: {row[4]}")
                print()


    def create(self):
        print(f"Creating a new {self.name} record...")
        while True:
            try:
                project_id = int(input(f"{self.name}_Id: "))
                break
            except:
```

```
            print(f"{self.name}_id must be Integer")
        name = input("Name: ")
        description = input("Description: ")
        while True:
            try:
                freelancer_id = int(input(f"Freelancer_Id: "))
                while True:
                    try:
                        client_id = int(input(f"Client_Id: "))
                        break
                    except:
                        print(f"Client_Id must be Integer")
                break
            except:
                print(f"Freelancer_Id must be Integer")
        sql_insert = f"INSERT INTO {self.name} (Project_Id, Name, Description, Freelancer_Id, Client_Id) VALUES (%s, %s, %s, %s, %s)"
        with self.conn, self.conn.cursor() as cur:
                cur.execute(sql_insert, (project_id, name, description, freelancer_id, client_id))


    def generate(self):
        print(f"Generating random {self.name} records...")
        while True:
            try:
                records_amount = int(input("How many records to generate?"))
                break
            except:
```

```
            print("Records amount must be Integer")
            sql_generate = f"INSERT INTO {self.name} (Project_Id, Name, Description, Freelancer_Id, Client_Id) (select {self.sqlRandomInteger}, {self.sqlRandomString}, {self.sqlRandomString}, {self.sqlRandomInteger}, {self.sqlRandomInteger})"
        for _ in range(records_amount):
            with self.conn, self.conn.cursor() as cur:
                cur.execute(sql_generate)
```

Task,py

```
from Controller import Controller

class Task(Controller):
    def read(self, where_condition=""):
        if not(where_condition): print(f"Reading a {self.name} records...")
            sql_select = f"SELECT Task_Id, Description, State, Project_Id FROM
{self.name} {where_condition}"
        with self.conn, self.conn.cursor() as cur:
            cur.execute(sql_select)
            for row in cur.fetchall():
                print(f"Task_Id: {row[0]}")
                print(f"Name: {row[1]}")
                print(f"State: {row[2]}")
                print(f"Project_Id: {row[3]}")
                print()

    def create(self):
        print(f"Creating a new {self.name} record...")
        while True:
            try:
                project_id = int(input(f"{self.name}_Id: "))
                break
            except:
                print(f"{self.name}_id must be Integer")
        description = input("Description: ")
```

```
        state = input("State: ")
        while True:
            try:
                freelancer_id = int(input(f"Project_Id: "))
                break
            except:
                print(f"Project_Id must be Integer")
            sql_insert = f"INSERT INTO {self.name} (Task_Id, Description, State,
Project_Id) VALUES (%s, %s, %s, %s, %s)"
        with self.conn, self.conn.cursor() as cur:
            cur.execute(sql_insert, (task_id, description, state, project_id))



    def generate(self):
        print(f"Generating random {self.name} records...")
        while True:
            try:
                records_amount = int(input("How many records to generate?"))
                break
            except:
                print("Records amount must be Integer")
            sql_generate = f"INSERT INTO {self.name} (Task_Id, Description, State,
Project_Id)        (select    {self.sqlRandomInteger},    {self.sqlRandomString},
{self.sqlRandomString}, {self.sqlRandomInteger})"
        for _ in range(records_amount):
            with self.conn, self.conn.cursor() as cur:
                cur.execute(sql_generate)
```

Client.py

```
from Controller import Controller

class Client(Controller):
    def read(self, where_condition=""):
        if not(where_condition): print(f"Reading a {self.name} records...")
            sql_select = f"SELECT Client_Id, Name, Contacts FROM {self.name}
{where_condition}"
        with self.conn, self.conn.cursor() as cur:
            cur.execute(sql_select)
            for row in cur.fetchall():
                print(f"Client_Id: {row[0]}")
                print(f"Name: {row[1]}")
                print(f"Contacts: {row[2]}")
                print()

    def create(self):
        print(f"Creating a new {self.name} record...")
        while True:
            try:
                client_id = int(input(f"{self.name}_Id: "))
                break
            except:
                print(f"{self.name}_id must be Integer")
        name = input("Name: ")
        contacts = input("Contacts: ")
```

```
            sql_insert = f"INSERT INTO {self.name} (Client_Id, Name, Contacts)
VALUES (%s, %s, %s)"
        with self.conn, self.conn.cursor() as cur:
            cur.execute(sql_insert, (client_id, name, contacts))



    def generate(self):
        print(f"Generating random {self.name} records...")
        while True:
            try:
                records_amount = int(input("How many records to generate?"))
                break
            except:
                print("Records amount must be Integer")
        sql_generate = f"INSERT INTO {self.name} (Client_Id, Name, Contacts)
(select {self.sqlRandomInteger}, {self.sqlRandomString},
{self.sqlRandomString})"
        for _ in range(records_amount):
            with self.conn, self.conn.cursor() as cur:
                cur.execute(sql_generate)
```