НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ
Кафедра системного програмування та спеціалізованих комп'ютерних систем

Розрахункова 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-11
Лабазов Володимир Володимирович
Перевірив: Петрашенко А.В.

Київ – 2023

Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL

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).

Опис предметної галузі з лабораторної роботи №1

При проектуванні бази даних «Система обліку екзаменаційних балів студентів» я виділив наступні сутності: Студент (Student), Група (Group), Предмет (Discipline), Оцінка (Mark).

Група може містити багато студентів , але студент може знаходитись в одній групі (зв'язок 1:N).

Кожен студент має багато оцінок (зв'язок 1:N).

Кожен студент ходить на не одну дисципліну , і на не одну дисципліну ходять не один студент (зв'язок N:M).

Кожна оцінка має один предмет з якого вона була отримана , і кожен предмет має одну оцінку за екзамен (зв'язок 1:1).

Таблиця сутностей з описом їх призначення

Сутність	Атрибут	Тип (розмір)
Сутність «Student» містить інформацію про студента	id (PK) – унікальний id студента name – ім’я студента group_id(FK) – унікальний id власника	Числовий Текстовий (50) Числовий
Сутність «Group» містить інформацію про групу	id (PK) – унікальний id групи name – назва групи	Числовий Текстовий (5)
Сутність «Discipline» містить інформацію про предмет	id (PK) – унікальний id дисципліни name – назва дисципліни teacher_name – ім’я викладача дисципліни	Числовий Текстовий (50) Текстовий (50)
Сутність «Mark» містить інформацію про екзаменаційну оцінку	value – оцінка в балах dicipline_id(FK) – id дисципліни з якої була отримана оцінка student_id(FK) – id студента , що отримав оцінку when_received – унікальний id головного тренера	Числовий Числовий Числовий Дата

Концептуальна модель предметної області “Система обліку екзаменаційних балів студентів”

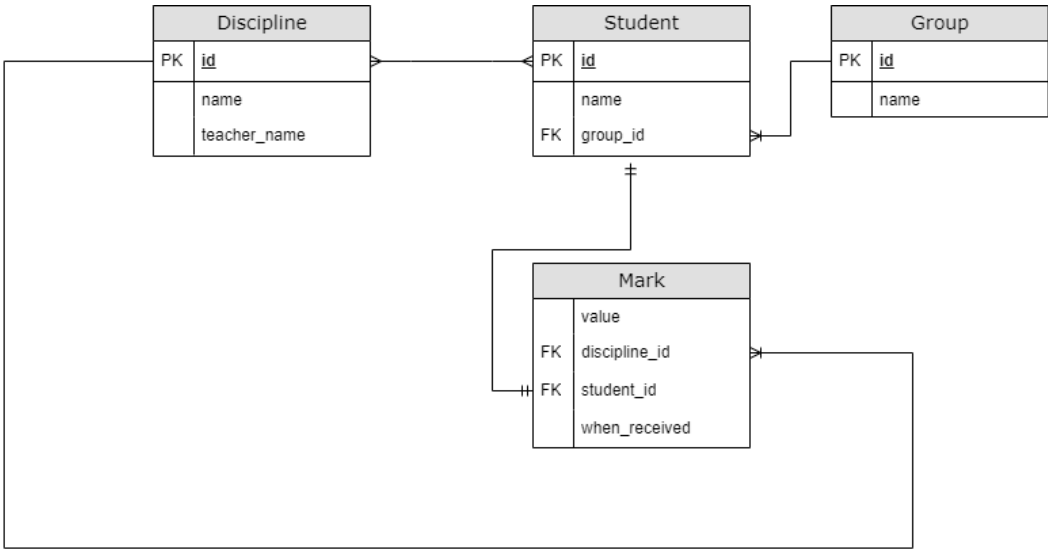
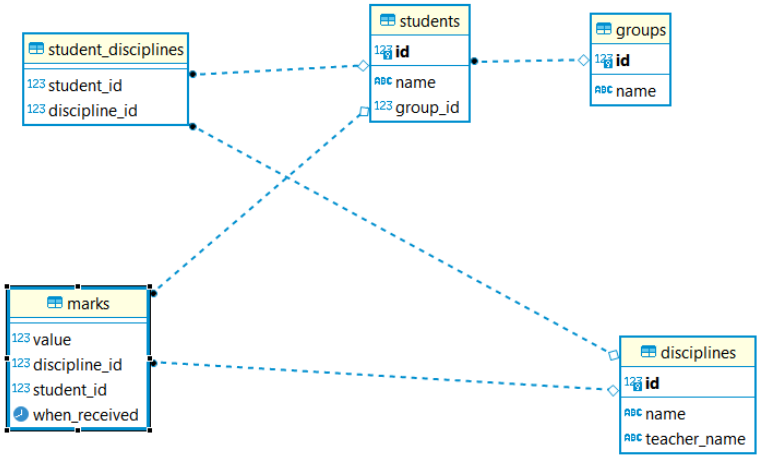


Рисунок 1 – Концептуальна модель предметної області «Система обліку екзаменаційних балів студентів».

Нотація: «UML». Модель побудована засобами програми draw.io

Опис процесу перетворення

Сутності «Student», «Group», «Discipline», «Mark» було перетворено у таблиці. Зв'язок між студентом та предметом (зв'язок багато до багатьох) зумовив появі додаткової таблиці «student_disciplines», яка містить унікальні id студента та дисципліни.



Код програми

Файл *crud.py*

```
from src.controller import Controller


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

Файл *controller.py*

```
from .model import Model
from .view import View
from functools import wraps
from psycopg2.errors import StringDataRightTruncation


def catch_db_error(option):
    @wraps(option)
    def inner(self, *args, **kwargs):
        try:
```

```
            option(self, *args, **kwargs)
        except (IndexError, StringDataRightTruncation,
ValueError, AssertionError):
            self.view.output_error_message()

    return inner


class Controller:
    def __init__(self):
        self.available = {
            "create": {
                "db": self.reset,
                "student": self.create_student,
                "group": self.create_group,
                "discipline": self.create_discipline,
                "mark": self.create_mark,
            },
            "read": {
                "students": self.read,
                "groups": self.read,
                "disciplines": self.read,
                "marks": self.read,
            },
            "update": {
                "student": self.update_student,
                "group": self.update_group,
                "discipline": self.update_discipline,
                "mark": self.update_mark,
            },
            "delete": {
                "student": self.delete_student,
                "groups": self.delete_group,
                "discipline": self.delete_discipline,
                "mark": self.delete_mark,
            },
            "requests": {
                "student_rating": self.request_rating,
                "avg_group_discipline_mark":
self.request_avg_mark,
                "student_group_list": self.request_group_list,
            },
        }
        self.model = Model()
```

```
        self.view = View()

    def run(self):
        while True:
            chosen_mode_viewer, chosen_mode =
self.view.show_menu()
            if not chosen_mode_viewer:
                self.model.disconnect()
                break
            chosen_option_viewer, chosen_option =
chosen_mode_viewer()
            args_or_command = chosen_option_viewer()
            self.available[chosen_mode][chosen_option]
(args_or_command)

    def reset(self, type_of_reset):
        if type_of_reset == "reset_fill":
            self.model.reset_db(True)
        else:
            self.model.reset_db(False)

    def request_rating(self, _):
        table = self.model.request_rating()
        self.view.output_table(table, "student_rating")

    def request_avg_mark(self, _):
        table = self.model.request_avg_mark()
        self.view.output_table(table,
"avg_group_discipline_mark")

    @catch_db_error
    def request_group_list(self, args):
        _, group_name = args
        table = self.model.request_group_list(group_name)
        self.view.output_table(table, "group_list")

    @catch_db_error
    def create_student(self, args):
        name, group_name = args
        self.model.create_student(name, group_name)

    def create_group(self, name):
        self.model.create_group(name)
```

```
    def create_discipline(self, args):
        name, teacher_name = args
        self.model.create_discipline(name, teacher_name)

    @catch_db_error
    def create_mark(self, args):
        value, mark_date, student_name, discipline_name = args
        self.model.create_mark(value, mark_date, student_name,
discipline_name)

    def read(self, read_from):
        table = self.model.read(read_from)
        self.view.output_table(table, read_from)

    @catch_db_error
    def update_student(self, args):
        name, what_to_change, new_value = args
        self.model.update_student(name, what_to_change,
new_value)

    @catch_db_error
    def update_group(self, args):
        name, new_value = args
        self.model.update_group(name, new_value)

    @catch_db_error
    def update_discipline(self, args):
        find_name, what_to_change, new_value = args
        self.model.update_discipline(find_name, what_to_change,
new_value)

    @catch_db_error
    def update_mark(self, args):
        find_student, find_discipline, what_to_change, new_value
= args
        self.model.update_mark(find_student, find_discipline,
what_to_change, new_value)

    @catch_db_error
    def delete_student(self, name):
        self.model.delete_student(name)

    @catch_db_error
    def delete_group(self, name):
```

```
        self.model.delete_group(name)

    @catch_db_error
    def delete_discipline(self, name):
        self.model.delete_discipline(name)

    @catch_db_error
    def delete_mark(self, args):
        student, discipline = args
        self.model.delete_mark(student, discipline)
```

Файл *model.py*

```
from .scripts.reset_db import reset
from psycopg2 import connect
from datetime import datetime


class Model:
    def __init__(self):
        self.connection = connect(
            database="students",
            user="admin",
            password="admin",
            host="127.0.0.1",
            port="5432",
        )
        self.get_id_queries = {
            "student": "select s.id\n"
                       "from students as s\n"
                       "where s.name = '{}'",
            "group": "select g.id\n"
                     "from groups as g\n"
                     "where g.name = '{}'",
            "discipline": "select d.id\n"
                          "from disciplines as d\n"
                          "where d.name = '{}'",
        }
        self.insert_queries = {
            "students": """INSERT INTO students(name, group_id)
VALUES (%s, %s)""",
            "groups": """INSERT INTO groups(name) VALUES
(%s)""",
            "disciplines": """INSERT INTO disciplines(name,
```

```
teacher_name) VALUES (%s, %s)""",
            "student_disciplines": """INSERT INTO
student_disciplines(student_id, discipline_id) VALUES (%s,
%s)""",
            "marks": """INSERT INTO marks(value, discipline_id,
student_id, when_received) VALUES (%s, %s, %s, %s)""",
        }
        self.read_queries = {
            "students": "select s.name, g.name as group_name\n"
                        "from students as s\n"
                        "inner join groups as g on s.group_id =
g.id",
            "groups": "select g.name as group_name\n"
                      "from groups as g",
            "disciplines": "select d.name as discipline_name,
d.teacher_name as teacher_name\n"
                           "from disciplines as d",
            "marks": "select "
                     "m.value as mark_value, "
                     "s.name as student_name, "
                     "d.name as discipline_name, "
                     "m.when_received as mark_date\n"
                     "from marks as m\n"
                     "inner join students as s on m.student_id =
s.id\n"
                     "inner join disciplines as d on
m.discipline_id = d.id",
        }

        self.request_queries = {
            "student_rating": "SELECT ROUND(AVG(m.value), 2) as
avg_mark, s.name\n"
                              "FROM marks as m\n"
                              "INNER JOIN students as s ON
m.student_id = s.id\n"
                              "GROUP BY s.id\n"
                              "ORDER BY avg_mark DESC",
            "avg_group_discipline_mark": "SELECT
ROUND(AVG(m.value), 2) as average_mark, "
                                         "g.name as group_name,
d.name as discipline_name\n"
                                         "FROM marks as m\n"
                                         "INNER JOIN students as
s ON m.student_id = s.id\n"
```

```
                                            "INNER JOIN disciplines
as d ON m.discipline_id = d.id\n"
                                            "INNER JOIN groups as g
ON s.group_id = g.id\n"
                                            "GROUP BY d.name,
g.name\n"
                                            "ORDER BY g.name",
            "student_group_list": "SELECT s.name as
student_name, g.name as group_name\n"
                                      "FROM students as s\n"
                                      "INNER JOIN groups as g ON
s.group_id = g.id\n"
                                      "WHERE g.id = {}\n",
        }

    def disconnect(self):
        if self.connection.closed == 0:
            self.connection.close()

    def _execute_select(self, request: str) -> list:
        cur = self.connection.cursor()
        cur.execute(request)
        return cur.fetchall()

    def _execute_insert(self, where_to_insert: str, data) ->
None:
        cur = self.connection.cursor()
        cur.execute(self.insert_queries[where_to_insert], data)
        self.connection.commit()
        cur.close()

    def _execute_query(self, query: str) -> None:
        cur = self.connection.cursor()
        cur.execute(query)
        self.connection.commit()
        cur.close()

    def reset_db(self, type_of_reset):
        reset(type_of_reset, self.connection)

    def request_rating(self):
        return
self._execute_select(self.request_queries["student_rating"])
```

```
    def request_avg_mark(self):
        return
self._execute_select(self.request_queries["avg_group_discipline_
mark"])

    def request_group_list(self, group_name):
        group_id =
self._execute_select(self.get_id_queries["group"].format(group_n
ame))[0][0]
        return
self._execute_select(self.request_queries["student_group_list"].
format(group_id))

    def create_student(self, student_name, group_name):
        group_id =
self._execute_select(self.get_id_queries["group"].format(group_n
ame))[0][0]
        prepared_data = ((student_name,), (group_id,))
        self._execute_insert("students", prepared_data)

    def create_group(self, group_name):
        prepared_data = ((group_name,),)
        self._execute_insert("groups", prepared_data)

    def create_discipline(self, discipline_name, teacher_name):
        prepared_data = ((discipline_name,), (teacher_name,))
        self._execute_insert("disciplines", prepared_data)

    def create_mark(self, mark_value, mark_date, student_name,
discipline_name):
        student_id =
self._execute_select(self.get_id_queries["student"].format(stude
nt_name))[0][0]
        discipline_id =
self._execute_select(self.get_id_queries["discipline"].format(di
scipline_name))[0][0]
        check_if_exists = self._execute_select(
            f"select sd.student_id\n"
            f"from student_disciplines as sd\n"
            f"where sd.discipline_id = {discipline_id}")
        if len(check_if_exists) == 0:
            self._execute_insert("student_disciplines",
((student_id,), (discipline_id,)))
        prepared_data = ((mark_value,), (discipline_id,),
```

```
(student_id,), (mark_date,))
        self._execute_insert("marks", prepared_data)

    def read(self, read_from):
        result =
self._execute_select(self.read_queries[read_from])
        return result

    def update_student(self, find_name, what_to_change,
new_value):
        student_id =
self._execute_select(self.get_id_queries["student"].format(find_
name))[0][0]
        value_to_set = new_value
        if what_to_change == "group_id":
            group_name = new_value
            value_to_set = self._execute_select(
                f"select g.id\n"
                f"from groups as g\n"
                f"where g.name = '{group_name}'"
            )[0][0]
        self._execute_query(
            f"update students\n"
            f"set {what_to_change} = '{value_to_set}'\n"
            f"where id = {student_id};"
        )

    def update_group(self, find_name, new_value):
        group_id =
self._execute_select(self.get_id_queries["group"].format(find_na
me))[0][0]
        self._execute_query(
            f"update groups\n"
            f"set name = '{new_value}'\n"
            f"where id = {group_id};"
        )

    def update_discipline(self, find_name, what_to_change,
new_value):
        discipline_id =
self._execute_select(self.get_id_queries["discipline"].format(fi
nd_name))[0][0]
        self._execute_query(
            f"update disciplines\n"
```

```
            f"set {what_to_change} = '{new_value}'\n"
            f"where id = {discipline_id};"
        )

    def update_mark(self, find_student, find_discipline,
what_to_change, new_value):
        student_id =
self._execute_select(self.get_id_queries["student"].format(find_
student))[0][0]
        discipline_id =
self._execute_select(self.get_id_queries["discipline"].format(fi
nd_discipline))[0][0]
        if what_to_change == "value":
            assert 1 <= int(new_value) <= 12
        elif what_to_change == "when_received":
            new_value = datetime.strptime(new_value, "%d.%m.
%Y").date()

        self._execute_query(
            f"update marks\n"
            f"set {what_to_change} = '{new_value}'\n"
            f"where student_id = '{student_id}' and
discipline_id = '{discipline_id}'"
        )

    def delete_student(self, name):
        student_id =
self._execute_select(self.get_id_queries["student"].format(name)
)[0][0]

        query = f"delete from student_disciplines where
student_id = {student_id};\n" \
                f"delete from marks where student_id =
{student_id};\n" \
                f"delete from students where id = {student_id};"
        self._execute_query(query)

    def delete_group(self, name):
        group_id =
self._execute_select(self.get_id_queries["group"].format(name))
[0][0]
        students = self._execute_select(
            f"select s.name\n"
            f"from students as s\n"
```

```
            f"where s.group_id = '{group_id}'"
        )
        students = [student[0].strip() for student in students]
        for s_name in students:
            self.delete_student(s_name)

        self._execute_query(f"delete from groups where id =
{group_id}")

    def delete_discipline(self, name):
        discipline_id =
self._execute_select(self.get_id_queries["discipline"].format(na
me))[0][0]

        query = f"delete from student_disciplines where
discipline_id = {discipline_id};\n" \
                f"delete from marks where discipline_id =
{discipline_id};\n" \
                f"delete from disciplines where id =
{discipline_id};"
        self._execute_query(query)

    def delete_mark(self, find_student, find_discipline):
        student_id =
self._execute_select(self.get_id_queries["student"].format(find_
student))[0][0]
        discipline_id =
self._execute_select(self.get_id_queries["discipline"].format(fi
nd_discipline))[0][0]

        query = \
            f"delete from student_disciplines where student_id =
{student_id} and discipline_id = {discipline_id};\n" \
            f"delete from marks where student_id = {student_id}
and discipline_id = {discipline_id};"
        self._execute_query(query)
```

Файл *view.py*

```
from typing import Callable, Union
from datetime import datetime
from tabulate import tabulate
```

```
class View:

    def __init__(self):
        self.available_commands_menus: dict = {
            "create": self.show_menu_create,
            "read": self.show_menu_read,
            "update": self.show_menu_update,
            "delete": self.show_menu_delete,
            "requests": self.show_menu_requests,
            "quit": None,
        }
        self.available_requests: dict = {
            "student_rating": self.show_requests_rating,
            "avg_group_discipline_mark":
self.show_requests_avg_mark,
            "student_group_list": self.show_requests_group_list,
        }
        self.available_create: dict = {
            "db": self.show_create_db,
            "student": self.show_create_student,
            "group": self.show_create_group,
            "discipline": self.show_create_discipline,
            "mark": self.show_create_mark,
        }
        self.available_read: dict = {
            "students": self.show_read_students,
            "groups": self.show_read_groups,
            "disciplines": self.show_read_disciplines,
            "marks": self.show_read_marks,
        }
        self.available_update: dict = {
            "student": self.show_update_students,
            "group": self.show_update_groups,
            "discipline": self.show_update_disciplines,
            "mark": self.show_update_marks,
        }
        self.available_delete: dict = {
            "student": self.show_delete_student,
            "groups": self.show_delete_group,
            "discipline": self.show_delete_discipline,
            "mark": self.show_delete_mark,
        }
        self.table_headers: dict = {
            "students": ("student_name", "group_name"),
```

```
            "groups": ("group_name", ),
            "disciplines": ("discipline_name", "teacher_name"),
            "marks": ("mark_value", "student_name",
"discipline_name", "mark_date"),
            "student_rating": ("avg_mark", "student_name"),
            "avg_group_discipline_mark": ("average_mark",
"group_name", "discipline_name"),
            "group_list": ("student_name", "group_name"),
        }

    def output_table(self, table, table_name):
        print("\n\n")
        print(
            tabulate(
                [[field.strip() if type(field) is str else field
for field in row] for row in table],
                headers=self.table_headers[table_name]
            )
        )

    @staticmethod
    def output_error_message():
        print("Incorrect input")

    @staticmethod
    def _output_options(options_dict: dict, amount_of_tabs: int,
title: str) -> None:
        options = tuple(options_dict.keys())
        tab_string = "\t" * amount_of_tabs
        print(f"\n\n{tab_string}{title}:\n")
        for index, option in enumerate(options):
            print(f"{tab_string}{index + 1}. {option}\n")

    @staticmethod
    def _handle_wrong_input(options: dict) -> Union[Callable,
str]:
        while True:
            try:
                keys = tuple(options.keys())
                option = keys[int(input("Input number of the
option: ").strip()) - 1]
                return options[option]
            except (IndexError, ValueError):
                print("No such option, try again")
```

```
    @staticmethod
    def _get_key_by_value(dct: dict, value):
        keys = tuple(dct.keys())
        values = tuple(dct.values())
        index = values.index(value)
        return keys[index]

    def show_menu(self) -> tuple[Callable, str]:
        self._output_options(
            self.available_commands_menus,
            amount_of_tabs=0,
            title="Choose one option from the options given
below"
        )
        response =
self._handle_wrong_input(self.available_commands_menus)
        return response,
self._get_key_by_value(self.available_commands_menus, response)

    def show_menu_requests(self) -> tuple[Callable, str]:
        self._output_options(
            self.available_requests,
            amount_of_tabs=1,
            title="Choose request"
        )
        response =
self._handle_wrong_input(self.available_requests)
        return response,
self._get_key_by_value(self.available_requests, response)

    @staticmethod
    def show_requests_rating() -> str:
        return "student_rating"

    @staticmethod
    def show_requests_avg_mark() -> str:
        return "avg_group_discipline_mark"

    @staticmethod
    def show_requests_group_list() -> tuple[str, str]:
        while True:
            group = input("Input group name:")
            if len(group) > 5:
```

```
                print("\nPlease input group name that fits in 5
characters\n")
                continue
            return "student_group_list", group

    def show_menu_create(self) -> tuple[Callable, str]:
        self._output_options(
            self.available_create,
            amount_of_tabs=1,
            title="Choose what do you want to create"
        )
        response =
self._handle_wrong_input(self.available_create)
        return response,
self._get_key_by_value(self.available_create, response)

    def show_create_db(self):
        options = {
            "reset": "reset",
            "reset_fill": "reset_fill",
        }
        self._output_options(options, amount_of_tabs=2,
title="Choose type of creating/reseting db")
        return self._handle_wrong_input(options)

    @staticmethod
    def show_create_student():
        while True:
            student = input("Input student name:")
            if len(student) > 50:
                print("\nPlease input student name that fits in
50 characters\n")
                continue
            else:
                break
        print("\n")
        group = input("Input student group:")
        return student, group

    @staticmethod
    def show_create_group():
        while True:
            group = input("Input group name:")
            if len(group) > 5:
```

```
                print("\nPlease input group name that fits in 5
characters\n")
                continue
            return group

    @staticmethod
    def show_create_mark():
        student_name = input("Input name of person, who received
this mark:")
        discipline_name = input("Input discipline name:")
        while True:
            try:
                mark = int(input("Input mark:"))
                assert 1 <= mark <= 12
                when_received = datetime.strptime(input("Input
when mark was received:"), "%d.%m.%Y").date()
                return mark, when_received, student_name,
discipline_name
            except (ValueError, AssertionError):
                print("Please input correct value")

    @staticmethod
    def show_create_discipline():
        while True:
            discipline_name = input("Input name of the
discipline:")
            if len(discipline_name) > 50:
                print("\nPlease input discipline name that fits
in 50 characters\n")
                continue
            teacher_name = input("Input teacher name:")
            if len(teacher_name) > 50:
                print("\nPlease input teacher name that fits in
50 characters\n")
                continue
            return discipline_name, teacher_name

    def show_menu_read(self):
        self._output_options(
            self.available_read,
            amount_of_tabs=1,
            title="Choose what do you want to read"
        )
        response = self._handle_wrong_input(self.available_read)
```

```
        return response,
self._get_key_by_value(self.available_read, response)

    @staticmethod
    def show_read_students():
        return "students"

    @staticmethod
    def show_read_groups():
        return "groups"

    @staticmethod
    def show_read_disciplines():
        return "disciplines"

    @staticmethod
    def show_read_marks():
        return "marks"

    def show_menu_update(self):
        self._output_options(
            self.available_update,
            amount_of_tabs=1,
            title="Choose what do you want to update"
        )
        response =
self._handle_wrong_input(self.available_update)
        return response,
self._get_key_by_value(self.available_update, response)

    def show_update_students(self):
        while True:
            student = input("Input student name:")
            if len(student) > 50:
                print("\nPlease input student name that fits in
50 characters\n")
                continue
            else:
                break
        change_options = {"change_name": "name", "change_group":
"group_id"}
        self._output_options(
            change_options,
            amount_of_tabs=2,
```

```
            title="Choose what do you want to change"
        )
        response = self._handle_wrong_input(change_options)
        new_value = input("\nInput new value:")
        return student, response, new_value

    @staticmethod
    def show_update_groups():
        while True:
            group = input("Input group name:")
            if len(group) > 5:
                print("\nPlease input group name that fits in 5
characters\n")
                continue
            else:
                break
        new_value = input("\nInput new value:")
        return group, new_value

    def show_update_disciplines(self):
        while True:
            discipline_name = input("Input name of the
discipline:")
            if len(discipline_name) > 50:
                print("\nPlease input discipline name that fits
in 50 characters\n")
                continue
            else:
                break
        change_options = {"change_discipline_name": "name",
"change_teacher_name": "teacher_name"}
        self._output_options(
            change_options,
            amount_of_tabs=2,
            title="Choose what do you want to change"
        )
        response = self._handle_wrong_input(change_options)
        new_value = input("\nInput new value:")
        return discipline_name, response, new_value

    def show_update_marks(self):
        student_name = input("Input name of person, who received
this mark:")
        discipline_name = input("Input discipline name:")
```

```
        change_options = {"change_mark_value": "value",
"change_mark_date": "when_received"}
        self._output_options(
            change_options,
            amount_of_tabs=2,
            title="Choose what do you want to change"
        )
        response = self._handle_wrong_input(change_options)
        new_value = input("\nInput new value:")
        return student_name, discipline_name, response,
new_value

    def show_menu_delete(self):
        self._output_options(
            self.available_delete,
            amount_of_tabs=1,
            title="Choose what do you want to delete"
        )
        response =
self._handle_wrong_input(self.available_delete)
        return response,
self._get_key_by_value(self.available_delete, response)

    @staticmethod
    def show_delete_student():
        while True:
            student = input("Input student name:")
            if len(student) > 50:
                print("\nPlease input student name that fits in
50 characters\n")
                continue
            return student

    @staticmethod
    def show_delete_group():
        while True:
            group = input("Input group name:")
            if len(group) > 5:
                print("\nPlease input group name that fits in 5
characters\n")
                continue
            return group

    @staticmethod
```

```
    def show_delete_discipline():
        while True:
            discipline = input("Input discipline name:")
            if len(discipline) > 50:
                print("\nPlease input discipline name that fits
in 50 characters\n")
                continue
            return discipline

    @staticmethod
    def show_delete_mark():
        student_name = input("Input name of person, who received
this mark:")
        discipline_name = input("Input discipline name:")
        return student_name, discipline_name
```

Файл *scripts/reset_db.py*

```
from datetime import date
from faker import Faker
from random import randint, choice
from pathlib import Path

STUDENTS_AMOUNT = 6
GROUPS_AMOUNT = 3
DISCIPLINES_AMOUNT = 5
TEACHERS_AMOUNT = 3
EACH_STUDENT_MARKS_AMOUNT = 5
choose_discipline = [
    'Physics',
    'English',
    'Programming',
    'Data Structures and Algorithms',
    'Computer Logic',
    'Math Analysis',
    'History of science and technology',
    'Discrete Math',
    'Analytical geometry and linear algebra',
]


def create(connection) -> None:
    filepath = Path(__file__).parent.resolve() /
Path("all_marks.sql")
```

```
    with open(filepath, 'r') as file:
        scrypt = file.read()

    with connection.cursor() as cur:
        cur.execute(scrypt)

    cur.close()
    connection.commit()


def generate_data() -> tuple[list, list, list, list, list[list,
list]]:
    student_names = []
    group_names = []
    discipline_names = []
    teachers_names = []
    marks_and_dates = [[], []]
    fake_data = Faker('uk_UA')

    for _ in range(STUDENTS_AMOUNT):
        student_names.append(fake_data.name())

    for _ in range(TEACHERS_AMOUNT):
        teachers_names.append(fake_data.name())

    for _ in range(GROUPS_AMOUNT):
        group_names.append(fake_data.bothify(text='??-##'))

    for _ in range(DISCIPLINES_AMOUNT):
        discipline = choice(choose_discipline)
        discipline_names.append(discipline)
        choose_discipline.remove(discipline)

    for _ in range(EACH_STUDENT_MARKS_AMOUNT * STUDENTS_AMOUNT):
        marks_and_dates[0].append(randint(1, 12))
        marks_and_dates[1].append(
            fake_data.date_between(
                start_date=date(year=2023, month=1, day=5),
                end_date=date(year=2023, month=1, day=20),
            )
        )

    return student_names, group_names, discipline_names,
teachers_names, marks_and_dates
```

```
def prepare_data(student_names, group_names, discipline_names,
teachers_names, marks_and_dates) -> tuple:
    prepared_groups = []
    for group in group_names:
        prepared_groups.append((group,))

    prepared_students = []
    for student in student_names:
        prepared_students.append((student, randint(1,
GROUPS_AMOUNT)))

    prepared_disciplines = []
    for discipline in discipline_names:
        prepared_disciplines.append((discipline,
choice(teachers_names)))

    prepared_discipline_student_relationships = []
    for student_id in range(1, STUDENTS_AMOUNT + 1):
        disciplines_ids = list(range(1, DISCIPLINES_AMOUNT + 1))
        for d_id in disciplines_ids:

prepared_discipline_student_relationships.append((student_id,
d_id))

    prepared_marks = []
    discipline_student_marks_relationships_dict = {}
    for student_id in range(1, STUDENTS_AMOUNT + 1):
        discipline_student_marks_relationships_dict[student_id]
= []
    for student_discipline_ids in
prepared_discipline_student_relationships:
        discipline_student_marks_relationships_dict[
            student_discipline_ids[0]
        ].append(student_discipline_ids)
    for student, discipline in
discipline_student_marks_relationships_dict.items():
        for d_id in discipline:
            mark = marks_and_dates[0].pop(0)
            mark_date = marks_and_dates[1].pop(0)
            prepared_marks.append((mark, d_id[1], student,
mark_date))
```

```
    return (
        prepared_students,
        prepared_groups,
        prepared_disciplines,
        prepared_discipline_student_relationships,
        prepared_marks,
    )


def insert_data_to_db(
        students_table,
        groups_table,
        disciplines_table,
        student_disciplines_table,
        marks_table,
        connection
) -> None:
    cur = connection.cursor()

    sql_to_groups = """INSERT INTO groups(name) VALUES (%s)"""
    cur.executemany(sql_to_groups, groups_table)

    sql_to_students = """INSERT INTO students(name, group_id)
VALUES (%s, %s)"""
    cur.executemany(sql_to_students, students_table)

    sql_to_disciplines = """INSERT INTO disciplines(name,
teacher_name) VALUES (%s, %s)"""
    cur.executemany(sql_to_disciplines, disciplines_table)

    sql_to_student_disciplines = """INSERT INTO
student_disciplines(student_id, discipline_id) VALUES (%s,
%s)"""
    cur.executemany(sql_to_student_disciplines,
student_disciplines_table)

    sql_to_marks = """INSERT INTO marks(value, discipline_id,
student_id, when_received) VALUES (%s, %s, %s, %s)"""
    cur.executemany(sql_to_marks, marks_table)

    connection.commit()
    cur.close()
```

```
def reset(fill, connection) -> None:
    create(connection)
    if fill:
        students, groups, disciplines, teachers, marks =
generate_data()
        for_students, for_groups, for_disciplines,
for_student_disciplines, for_marks = prepare_data(
            students,
            groups,
            disciplines,
            teachers,
            marks
        )
        insert_data_to_db(for_students, for_groups,
for_disciplines, for_student_disciplines, for_marks, connection)
```

Файл *scripts/all_marks.sql*

```
DROP TABLE IF EXISTS groups CASCADE;
CREATE TABLE groups (
    id SERIAL PRIMARY KEY,
    name CHAR(5) UNIQUE NOT NULL
);

DROP TABLE IF EXISTS students CASCADE;
CREATE TABLE students (
    id SERIAL PRIMARY KEY,
    name CHAR(50) NOT NULL,
    group_id INTEGER,
    FOREIGN KEY (group_id) REFERENCES groups (id)
);

DROP TABLE IF EXISTS disciplines CASCADE;
CREATE TABLE disciplines (
    id SERIAL PRIMARY KEY,
    name CHAR(50) UNIQUE NOT NULL,
    teacher_name CHAR(50) NOT NULL
);

DROP TABLE IF EXISTS student_disciplines;
CREATE TABLE student_disciplines (
    student_id INTEGER,
    discipline_id INTEGER,
    FOREIGN KEY (student_id) REFERENCES students (id),
```

```
    FOREIGN KEY (discipline_id) REFERENCES disciplines (id)
);

DROP TABLE IF EXISTS marks;
CREATE TABLE marks (
    value SMALLINT NOT NULL,
    discipline_id INTEGER,
    student_id INTEGER,
    when_received DATE NOT NULL,
    FOREIGN KEY(discipline_id) REFERENCES disciplines (id),
    FOREIGN KEY(student_id) REFERENCES students (id)
);
```

Демонстрація роботи програми

Головне меню користувача

```
"C:\Python 3.9.6\python.exe" C:/UniverRepos/db_university/rgr/crud.py


Choose one option from the options given below:

1. create

2. read

3. update

4. delete

5. requests

6. quit

Input number of the option:
```

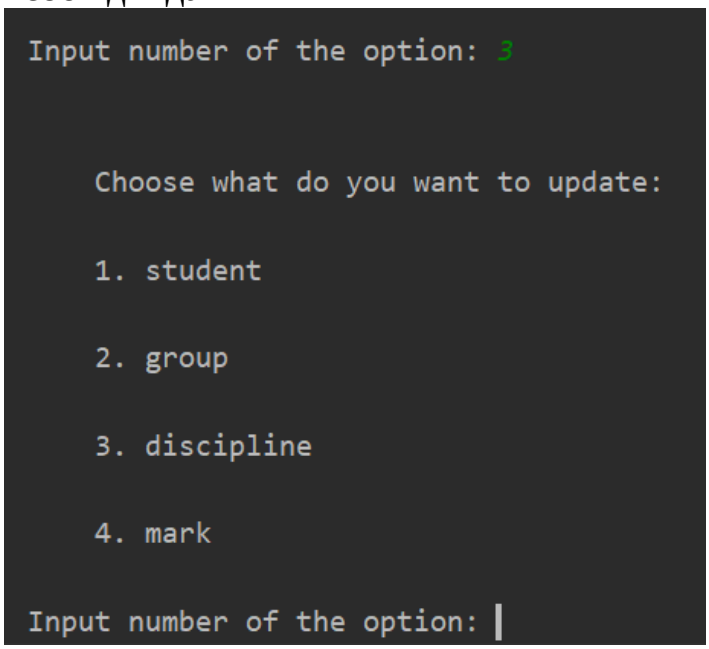
Меню створення
Через першу опцію можна видалити всі записи з бд, або перезаповнити її випадково згенерованими даними
Для створення запису в певній таблиці треба вибрати відповідну опцію і ввести необхідні дані

```
6. quit

Input number of the option: 1


    Choose what do you want to create:

    1. db

    2. student

    3. group

    4. discipline

    5. mark

Input number of the option:
```

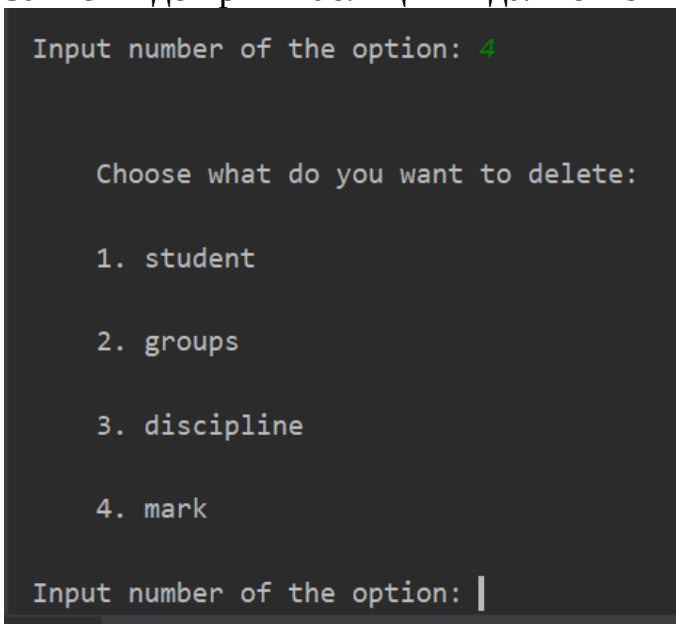
Меню читання
Тут потрібно просто вибрати необхідну опцію для показу вмісту певної таблиці

```
Input number of the option: 2


    Choose what do you want to read:

    1. students

    2. groups

    3. disciplines

    4. marks

Input number of the option: 2



group_name
------------
iJ-74
ra-87
eY-68
```

Меню модифікації
Для модифікації необхідно вибрати таблицю для модифікації і ввести необхідні дані



Меню видалення
Для видалення необхідно вибрати таблицю, з якої будуть видалятись дані, і ввести необхідні дані
Записи в дочірніх таблицях видаляються каскадно



Меню запитів

```
Input number of the option: 5


    Choose request:

    1. student_rating

    2. avg_group_discipline_mark

    3. student_group_list

Input number of the option: 1



  avg_mark  student_name
----------  -----------------
       8.2  В’ячеслав Скиба
       6.8  Данна Венгринович
       6.6  Михайло Сомко
       5.6  Василь Шило
       5.2  Онисим Худоб’як
       5    Тимофій Вишиваний
```

Помилки у всьому додатку відловлюються на різних етапах і виводяться відповідні помилкові повідомлення