

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 ІII 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”

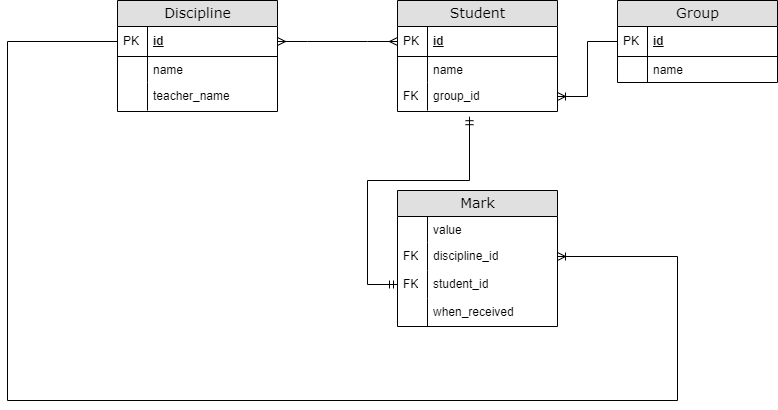


Рисунок 1 – Концептуальна модель предметної області «Система обліку екзаменаційних балів студентів».

**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\_name))[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\_name))[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(student\_name))[0][0]  
 discipline\_id = self.\_execute\_select(self.get\_id\_queries["discipline"].format(discipline\_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\_name))[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(find\_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(find\_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(name))[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(find\_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***

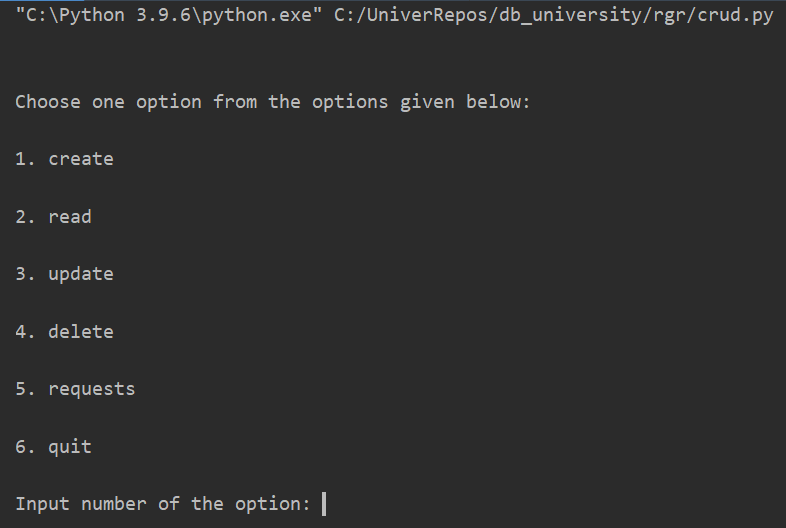
from datetime import date  
from faker import Faker  
from random import randint, choice  
from pathlib import Path  
  
STUDENTS\_AMOUNT = 6  
GROUPS\_AMOUNT = 3  
DISCIPLINES\_AMOUNT = 5  
TEACHERS\_AMOUNT = 3  
EACH\_STUDENT\_MARKS\_AMOUNT = 5  
choose\_discipline = [  
 'Physics',  
 'English',  
 '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**

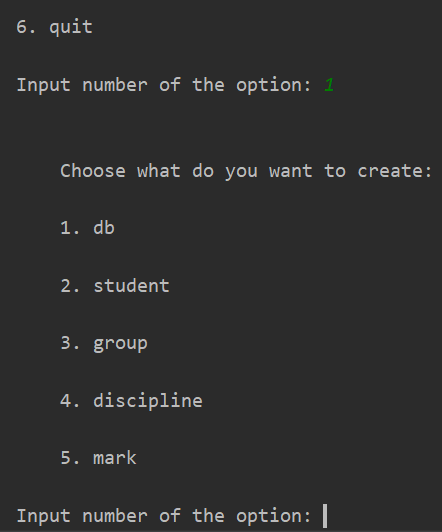
Головне меню користувача



Меню створення

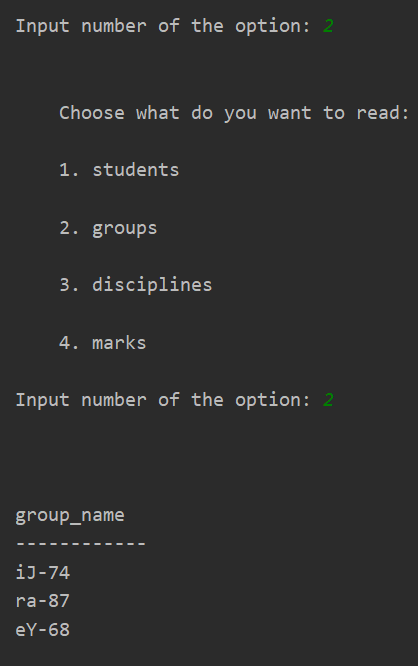
Через першу опцію можна видалити всі записи з бд, або перезаповнити її випадково згенерованими даними

Для створення запису в певній таблиці треба вибрати відповідну опцію і ввести необхідні дані



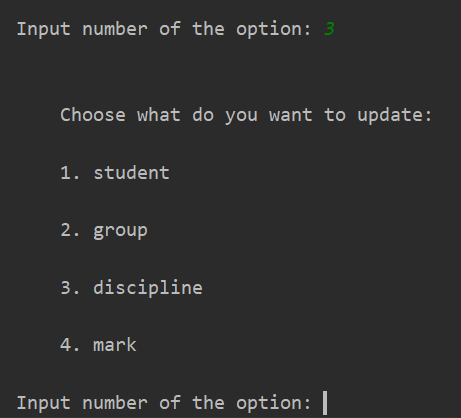
Меню читання

Тут потрібно просто вибрати необхідну опцію для показу вмісту певної таблиці



Меню модифікації

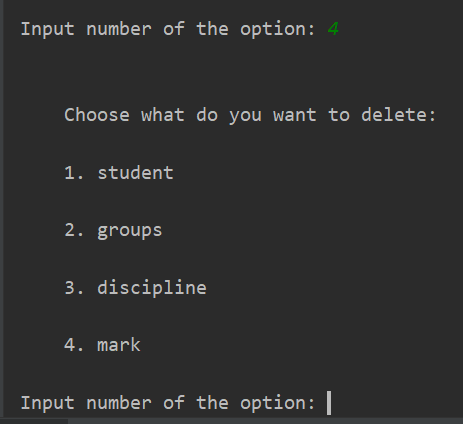
Для модифікації необхідно вибрати таблицю для модифікації і ввести необхідні дані



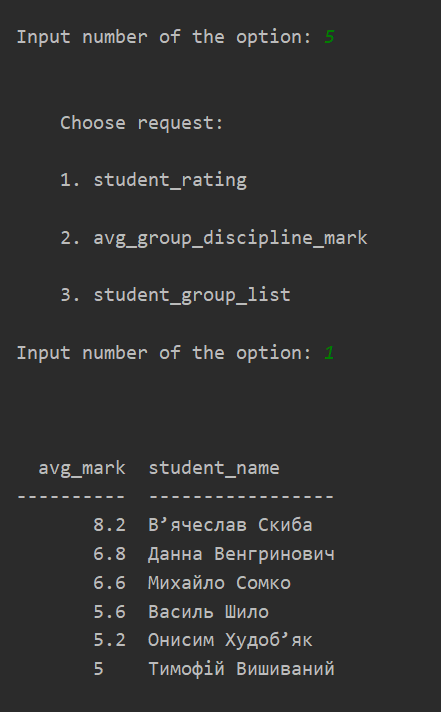
Меню видалення

Для видалення необхідно вибрати таблицю, з якої будуть видалятись дані, і ввести необхідні дані

Записи в дочірніх таблицях видаляються каскадно



Меню запитів



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