



НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування та спеціалізованих комп'ютерних систем

Лабораторна робота №2
з дисципліни
«Бази даних і засоби управління»

Виконав студент III курсу
ФПМ групи КВ-83
Пащенко Антон КВ-83
Перевірів: Павловський В.І.

Київ – 2020

Ознайомлення з базовими операціями СУБД PostgreSQL

Загальне завдання роботи полягає у наступному:

1. Реалізувати функції 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 2-х 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.

Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

Вимоги до інтерфейсу користувача

1. Використовувати консольний інтерфейс користувача.

Нормалізована логічна модель даних БД «Відеохостінг»

На Рис. Наведено нормалізовану логічну модель даних БД «Відеохостінг»

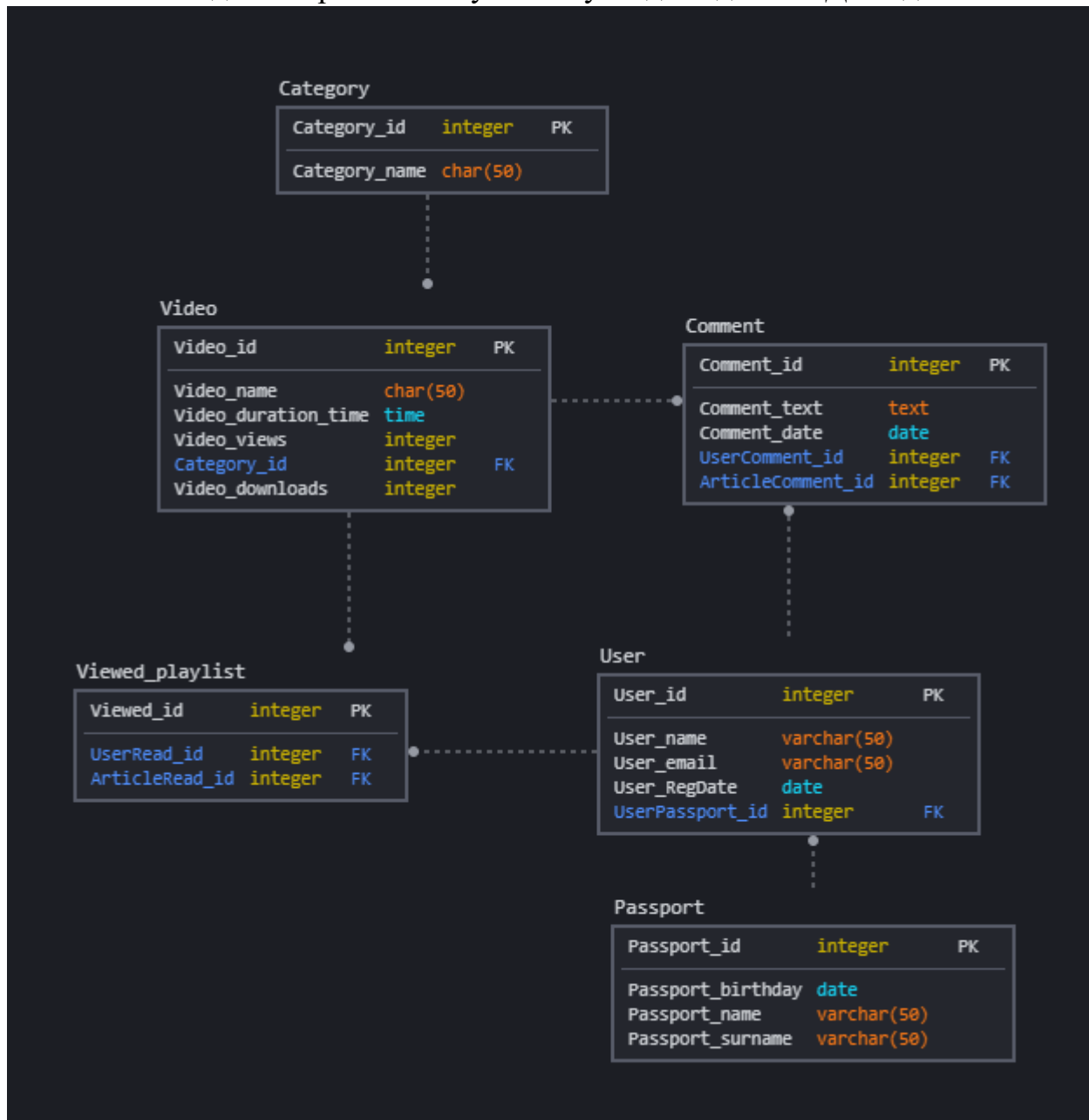


Рис 2.1 Нормалізована логічна модель даних БД «Відеохостінг»

Опис програми

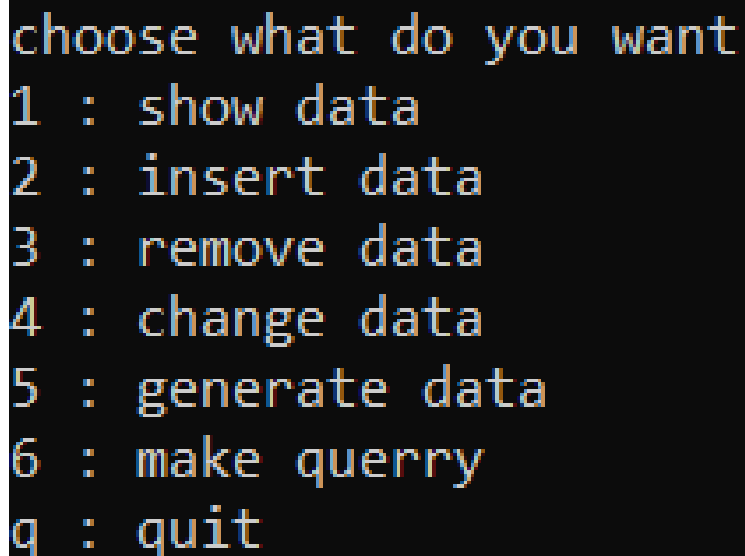
Програма створена для управління базою даних за допомогою базових операцій СУБД PostgreSQL та реалізовує функціональні вимоги, що наведені у завданні. Програма складається з 3 модулів:

1. `console_view` – клас що відповідає за відображення даних та обробку вводу користувача.
2. `controller.py` – клас що обробляє запити від `console_view` та викликає методи моделі.
3. `db_model` – клас що виконує запити до PostgreSQL та повертає результат.

Використані сторонні бібліотеки `psycopg2`, `prettytable`.

Структура меню програми

Меню програми



```
choose what do you want
1 : show data
2 : insert data
3 : remove data
4 : change data
5 : generate data
6 : make query
q : quit
```

Рис 2.2 Меню програми

Функціонал

Кожне меню запитує таблицю з якою буде проведена якась дія та додаткову інформацію в залежності від дії. При введенні неіснуючої таблиці програма повідомить про помилку та дасть можливість ввести коректні данні.

1. Show data виводить стовпці таблиці.
2. Insert data запрошує данні для конкретної таблиці, при введенні некоректних даних (за типом) програма повідомить про помилку та дасть можливість ввести коректні данні. Після успішного вводу кортеж додається до таблиці.
3. Remove data запрошує умову за якою будуть видалятися дані, при введенні некоректної умови програма повідомить про помилку та дасть можливість ввести коректну умову. Після успішного вводу данні що підходять під умову видаляються з таблиці.
4. Change data запрошує данні для конкретної таблиці на які треба змінити існуючі стовпці. Далі запрошується умова за якою будуть вибиратися кортежі що мають змінитися. При введенні некоректних даних (за типом або за умовою) програма повідомить про помилку та дасть можливість ввести коректні данні.
5. Generate data запрошує кількість кортежів, що треба згенерувати. При успішній генерації виводиться повідомлення.
6. Make query запрошує в якій комбінації таблиць буде проведений пошук video + category, users + passport, comment + video + users. Потім запрошується умова за якою буде проведений пошук. Якщо всі данні коректні виводиться результат інакше виводиться помилка і пропонується ввести нові данні.

Вибірка елементів з БД

Приклад вибірки елементів з таблиці users:

users				
user_id	user_name	user_email	user_regdate	user_passport_id
1	jay_querry	jay@gmail.com	2019-09-14	1
2	lucasito	lucas11@gmail.com	2015-03-01	2
4	ego4ik	egorich@gmail.com	2014-09-17	4
8	OH	TA	2015-11-12	8
9	DA	CG	2019-12-04	9
10	RP	GU	2014-02-15	10
11	TW	VO	2016-05-09	11

Рис 2.3 таблиця users

Метод що повертає вміст таблиці

```
def get_table_data(self, table_name):  
  
    id_column = self.get_column_types(table_name)[0][0]  
  
    cursor = self.__cursor  
    try:  
        cursor.execute(sql.SQL('SELECT * FROM {} ORDER BY {} ASC').format(sql.Identifier(table_name), sql.SQL(id_column)))  
    except Exception as e:  
        return str(e)  
  
    return ( [col.name for col in cursor.description] , cursor.fetchall())
```

module psycopg2.sql

Рис 2.4 Метод get_table_data

Видалення зв'язаних між собою даних

Наприклад таблиця Users з'єднана з таблицею Passport і Video то при видаленні запису з users будуть видаленні всі пов'язані записи з таблиць passport та video. Це реалізується засобами PostgreSQL за допомогою каскадного видалення.

Метод що видаляє данні з таблиці

```
def delete_data(self, table_name, cond):  
    self.__cursor.execute(  
        sql.SQL('DELETE FROM {} WHERE {}')  
        .format(sql.Identifier(table_name), sql.SQL(cond)))  
    self.__context.commit()
```

Рис 2.4 Метод delete_data

Структура програми

Програма створена за патерном MVC (Model-View-Controller). Складається з модулів Controller(controller.py), Model(db_model.py) View(console_view.py).

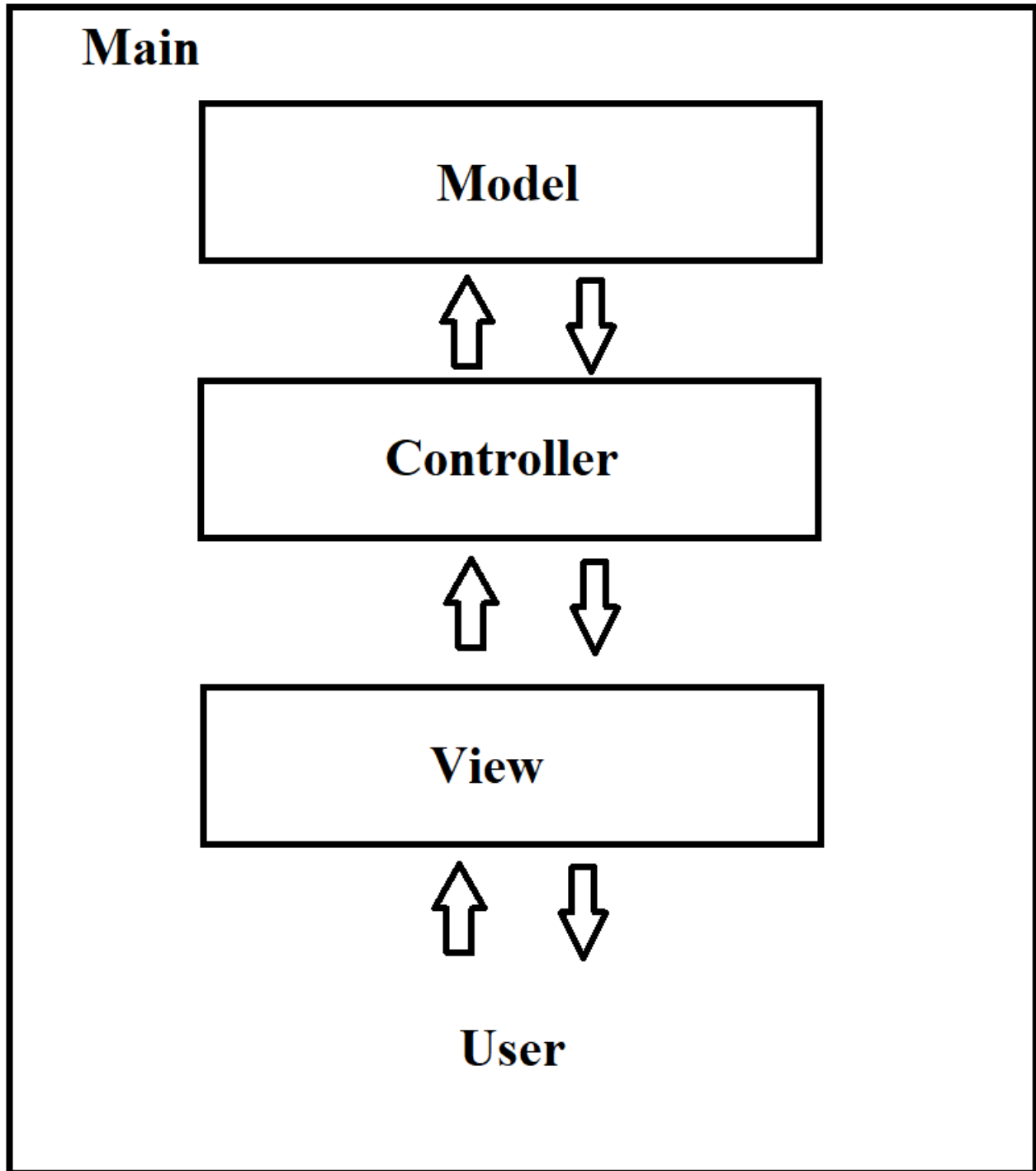


Рис 2.5 Структура програми

Додавання запису в таблицю

```
def insert_data(self, table_name, values):
    line = ''
    columns = '('
    for key in values:
        if values[key]:
            line += '%(' + key + ')s,'
            columns += key + ','

    columns = columns[:-1] + ')'
```

Рис 2.6 Метод insert_data

Додавання N згенерованих записів в таблицю

```
def generate_data(self, table_name, count):

    types = self.get_column_types(table_name)
    fk_array = self.get_foreign_key_info(table_name)
    select_subquery = ""
    insert_query = "INSERT INTO " + table_name + " ("
    for i in range(1, len(types)):
        t = types[i]
        name = t[0]
        type = t[1]
        fk = [x for x in fk_array if x[0] == name]
        if fk:
            select_subquery += '(SELECT {} FROM {} ORDER BY RANDOM(),ser LIMIT 1)'.format(fk[0][2], fk[0][1])
        elif type == 'integer':
            select_subquery += 'trunc(random()*100)::INT'
        elif type == 'character varying':
            select_subquery += 'chr(trunc(65 + random()*25)::INT) || chr(trunc(65 + random()*25)::INT)'
        elif type == 'date':
            select_subquery += "date(timestamp '2014-01-10' +
                                random() *
                                (timestamp '2020-01-20' - timestamp '2014-01-10'))"
        elif type == 'time without time zone':
            select_subquery += "time '00:00:00' + DATE_TRUNC('second',RANDOM() * time '24:00:00')"
        else:
            continue

    insert_query += name
    if i != len(types) - 1:
        select_subquery += ','
        insert_query += ','
    else:
        insert_query += ')'

    self.__cursor.execute(insert_query + "SELECT " + select_subquery + "FROM generate_series(1," + str(count) + ") as ser")
    self.__context.commit()
```

Рис 2.7 Метод generate_data

Зміна даних в таблиці

```
def change_data(self, table_name, values):
    line = ''
    condition = values.pop('condition')
    for key in values:
        if values[key]:
            line += key + '=%(' + key + ')s, '

    self.__cursor.execute(
        sql.SQL('UPDATE {} SET ' + line[:-1] + ' WHERE {} ')
        .format(sql.Identifier(table_name), sql.SQL(condition)),
        values)
    self.__context.commit()
```

Рис 2.8 Метод change_data

Пошук в декількох таблицях

```
def join_general(self, main_query, condition = ""):
    new_cond = condition
    if condition:
        new_cond = "WHERE " + condition

    t1 = time.time()
    self.__cursor.execute(main_query.format(new_cond))
    t2 = time.time()
    return ((t2 - t1) * 1000, self.__cursor.fetchall())

def join_video_category(self, condition = ""):
    return self.join_general("""SELECT * FROM video as v JOIN category as c ON v.video_category = c.category_id {} ORDER BY video_id ASC""", condition)

def join_users_passport(self, condition = ""):
    return self.join_general("""SELECT *
        FROM users as u JOIN passport as p ON u.user_passport_id = p.passport_id {} ORDER BY user_id ASC""", condition)

def join_comment_video_users(self, condition = ""):
    return self.join_general("""SELECT * FROM commentaries as c
        JOIN video as v ON c.comment_to_video = v.video_id
        JOIN users as u ON c.comment_from_user = u.user_id
        {}
        ORDER BY comment_id ASC""", condition)
```

Рис 2.9 Методи join_general, join_video_category, join_users_passport, join_comment_video_users

Приклад пошуку

WHERE user_id<5

user_id	user_name	user_email	user_regdate	user_passport_id	passport_id	passport_name	passport_surname	passport_date
1	jay_querry	jay@gmail.com	2019-09-14	1	1	Jay	Querry	2002-05-09
2	lucasito	lucas11@gmail.com	2015-03-01	2	2	Lucas	Bucky	1995-03-12
4	ego4ik	egorich@gmail.com	2014-09-17	4	4	Egor	Vasyliiev	1990-03-09

execution time 3.0014514923095703 ms

Рис 2.8 Об'єднані таблиці users та passport

Контроль на вставку даних в дочірню таблицю, коли нема відповідного запису в батьківській таблиці

```
INSERT MENU
print data to insert into users
user_name ( character varying ): Tes
user_email ( character varying ): Tes
user_regdate ( date ): 20-02-2020
user_passport_id ( integer ): 15
ОШИБКА: INSERT или UPDATE в таблице "users" нарушает ограничение внешнего ключа "user_passport_idfk"
DETAIL: Ключ (user_passport_id)=(15) отсутствует в таблице "passport".
```

Рис 2.9 Помилка при введення некоректного зовнішнього ключа

Код модуля console_view

```
import msvcrt # windows only

class console_view():
    def __init__(self, model, controller):
        self.__is_running = True;
        self.__model = model
        self.__controller = controller

    def quit(self):
        return None

    def print_message(self, message, callback = None):
        print(message)
        if callback is not None:
            return callback
        return None

    def print_table(self, table, callback = None):

        t = PrettyTable(table[0])
        for row in table[1]:
            t.add_row(row)
        print(t)

        if callback is not None:
            return callback
        return None

    def get_input(self, callback = None):
        inp = input()
```

```

        if callback is not None:
            pass

    def main_menu(self):
        print('choose what do you want')
        print('1 : show data')
        print('2 : insert data')
        print('3 : remove data')
        print('4 : change data')
        print('5 : generate data')
        print('6 : make query')
        print('q : quit')
        char = chr(msvcrt.getch()[0])
        return self.__controller.main_menu(char)

    def show_menu(self):
        print("SHOW MENU")
        table_names = self.__model.get_table_names()
        print('type table name or quit')
        print('suppored tables')
        print(table_names)
        answer = input()
        return self.__controller.show_menu(answer)

    def choose_delete_menu(self):
        print("DELETE MENU")
        table_names = self.__model.get_table_names()
        print('type table name','suppored tables',table_names)
        answer = input()
        return self.__controller.choose_delete_menu(answer)

    def choose_insert_menu(self):
        print("INSERT MENU")
        table_names = self.__model.get_table_names()
        print('type table name','suppored tables',table_names)
        answer = input()
        return self.__controller.choose_insert_menu(answer)

    def choose_change_menu(self):
        print("CHANGE MENU")
        table_names = self.__model.get_table_names()
        print('type table name','suppored tables',table_names)
        answer = input()
        return self.__controller.choose_change_menu(answer)

```

```

def choose_generate_menu(self):
    print("GENERATE MENU")
    table_names = self.__model.get_table_names()
    print('type table name', 'suppored tables', table_names)
    answer = input()
    return self.__controller.choose_generate_menu(answer)

def choose_query_menu(self, query_list):
    print("QUERY MENU")
    print("choose query")
    for i in range(0, len(query_list)):
        print(i + 1, ":", query_list[i])
    answer = input()

    return self.__controller.choose_query_menu(answer)

def insert_row_menu(self, table_name, columns_data):
    print("INSERT MENU")
    print('print data to insert into', table_name)
    answer = {}

    for data in columns_data:
        print(data[0], '(', data[1], '):', end=' ')
        inp = input()
        answer.update({data[0] : inp})

    return self.__controller.insert_data(table_name, answer)

def change_row_menu(self, table_name, columns_data):
    print("CHANGE MENU")
    print('print data to change into', table_name)
    answer = {}

    for data in columns_data:
        print(data[0], '(', data[1], '):', end=' ')
        inp = input()
        answer.update({data[0] : inp})
    print('WHERE', end=' ')
    inp = input()
    answer.update({'condition' : inp})

    return self.__controller.change_data(table_name, answer)

```

```

def delete_row_menu(self,table_name,column_arr):
    print("DELETE MENU")
    print(table_name)
    print("columns")
    print(column_arr)
    #self.__model.get
    print("WHERE",end = " ")
    answer = input()
    return self.__controller.delete_data(table_name,answer)

def generate_size_menu(self,table_name):
    print("GENERATE MENU")
    print('print how many rows you want to generate')
    answer = input()
    return self.__controller.generate_data(table_name,answer)

def cond_query_menu(self,query_num):
    print("QUERY MENU")
    print("WHERE",end = " ")
    cond = input()
    return self.__controller.cond_query_menu(query_num,cond)

```

Код модуля controller

```

class controller():
    def __init__(self, model):
        self.__model = model
        self.__query_data = ["video + category","users + passport","comment + video + users"]

    def set_view(self,view):
        self.__view = view

    def get_view_func(self,name,*args):
        return lambda : getattr(self.__view, name)(*args)

    def check_input(self,data,prev_menu = "main_menu"):

        if data == ':main':
            return self.get_view_func("main_menu")

        if data == ':ret':
            return self.get_view_func(prev_menu)

```

```

        if data == ':q':
            return self.get_view_func("quit")

    def main_menu(self, answer):
        if answer == '1':
            return self.get_view_func("show_menu")
        if answer == '2':
            return self.get_view_func("choose_insert_menu")
        if answer == '3':
            return self.get_view_func("choose_delete_menu")
        if answer == '4':
            return self.get_view_func("choose_change_menu")
        if answer == '5':
            return self.get_view_func("choose_generate_menu")
        if answer == '6':
            return self.get_view_func("choose_query_menu", self.__query_data)
        if answer == 'q':
            return self.get_view_func("quit")

        return self.get_view_func("main_menu")

    def show_menu(self, data):

        res = self.check_input(data)
        if res is not None:
            return res

        model_data = self.__model.get_table_data(data)

        if type(model_data) == str:
            return

        self.get_view_func("print_message", model_data, self.get_view_func("show_menu"))

        return self.get_view_func("print_table", model_data, self.get_view_func("show_menu"))

    def check_table(self, table_name, curr_menu):

        if table_name not in self.__model.get_table_names():
            return self.get_view_func("print_message", "unknown table
{0}".format(table_name), self.get_view_func(curr_menu))

        return None

```

```

def choose_insert_menu(self,data):

    res = self.check_input(data)
    if res is not None:
        return res

    res = self.check_table(data,"choose_insert_menu")

    if res is not None:
        return res

    return
self.get_view_func("insert_row_menu",data,self.__model.get_column_types(data)[1:])

def choose_delete_menu(self,data):

    res = self.check_input(data)
    if res is not None:
        return res

    res = self.check_table(data,"choose_delete_menu")

    if res is not None:
        return res

    column_arr = self.__model.get_column_names(data)

    return self.get_view_func("delete_row_menu",data,column_arr)

def choose_change_menu(self,data):

    res = self.check_input(data)
    if res is not None:
        return res

    res = self.check_table(data,"choose_change_menu")

    if res is not None:
        return res

    return
self.get_view_func("change_row_menu",data,self.__model.get_column_types(data)[1:])

def choose_generate_menu(self,data):

```

```

        res = self.check_input(data)
        if res is not None:
            return res

        res = self.check_table(data,"choose_generate_menu")

        if res is not None:
            return res

        return self.get_view_func("generate_size_menu",data)

def choose_query_menu(self,data):
    res = self.check_input(data)
    if res is not None:
        return res

    int_data = 0
    res_func = None
    try:
        int_data = int(data)
        if int_data <= len(self.__query_data) and int_data > 0:
            res_func = self.get_view_func("cond_query_menu",int_data)
        else:
            self.get_view_func("print_message","uknown index
{}".format(int_data),self.get_view_func("choose_query_menu",self.__query_data))
    except Exception as e:
        res_func =
self.get_view_func("print_message",str(e),self.get_view_func("choose_query_menu",self.__query
_data))

    return res_func

def change_data(self,table :str ,new_data : dict):

    res = self.check_input(new_data,"choose_change_menu")
    if res is not None:
        return res

    message = " "

    try:
        self.__model.change_data(table,new_data)

```



```

        message = "Success"
    except Exception as e:
        message = str(e)
        self.__model.clear_transaction()

    column_data = self.__model.get_column_types(table)
    return
self.get_view_func("print_message",message,self.get_view_func("choose_change_menu"))

def insert_data(self,table :str ,data : dict):

    res = self.check_input(data,"choose_insert_menu")
    if res is not None:
        return res

    message = " "
    try:
        self.__model.insert_data(table,data)
        message = "Success"
    except Exception as e:
        message = str(e)
        self.__model.clear_transaction()

    return
self.get_view_func("print_message",message,self.get_view_func("choose_insert_menu"))

def delete_data(self,table :str ,data : str):
    res = self.check_input(data,"choose_delete_menu")
    message = " "
    if res is not None:
        return res
    try:
        self.__model.delete_data(table,data)
        message = "Success"
    except Exception as e:
        message = str(e)
        self.__model.clear_transaction()

    column_arr = self.__model.get_column_names(table)

    return
self.get_view_func("print_message",message,self.get_view_func("delete_row_menu",table,column_
arr))

```

```

def generate_data(self,table_name,data):
    res = self.check_input(data,"choose_generate_menu")

    if res is not None:
        return res

    message = " "
    data_int = 0

    try:
        data_int = int(data)
    except Exception as e:
        return

self.get_view_func("print_message",str(e),self.get_view_func("generate_size_menu",table_name)
)

    if res is not None:
        return res

    if table_name == 'users':
        try:
            self.__model.generate_data_for_users(data_int)
            message = "Success"
        except Exception as e:
            message = str(e)
            self.__model.clear_transaction()
    else:
        try:
            self.__model.generate_data(table_name,data_int)
            message = "Success"
        except Exception as e:
            message = str(e)
            self.__model.clear_transaction()

    return

self.get_view_func("print_message",message,self.get_view_func("generate_size_menu",table_name
))

def cond_query_menu(self,query_num,cond):

    if cond == ':main':
        return self.get_view_func("main_menu")

```

```

        if cond == ':ret':
            return self.get_view_func("choose_query_menu",self.__query_data)

        if cond == ':q':
            return self.get_view_func("quit")

        query_func = None
        column_arr = []
        if query_num == 1:
            query_func = getattr(self.__model, "join_video_category")
            column_arr = self.__model.get_column_names('video') +
self.__model.get_column_names('category')
        elif query_num == 2:
            query_func = getattr(self.__model, "join_users_passport")
            column_arr = self.__model.get_column_names('users') +
self.__model.get_column_names('passport')
        elif query_num == 3:
            query_func = getattr(self.__model, "join_comment_video_users")
            column_arr = self.__model.get_column_names('commentaries')
            + self.__model.get_column_names('video')
            + self.__model.get_column_names('users')

        ret_func = None
        try:
            data = query_func(cond)
            ret_func = self.get_view_func("print_table", (column_arr,data[1]),
                                            self.get_view_func("print_message","execution time
{} ms".format(data[0]),self.get_view_func("cond_query_menu",query_num)))
        except Exception as e:
            ret_func =
self.get_view_func("print_message",str(e),self.get_view_func("cond_query_menu",query_num))
            self.__model.clear_transaction()

        return ret_func

```

Код модуля db_model

```

import psycopg2
from psycopg2 import sql
import time

class db_model():

```

```

def __init__(self, dbname, user_name, password, host):
    self.__context = psycopg2.connect(dbname=dbname, user=user_name,
                                      password=password, host=host)
    self.__cursor = self.__context.cursor()
    self.__table_names = None
    return None

def __del__(self):
    self.__cursor.close()
    self.__context.close()

def clear_transaction(self):
    self.__context.rollback()

def get_foreign_key_info(self, table_name):
    self.__cursor.execute("""
SELECT
    kcu.column_name,
    ccu.table_name AS foreign_table_name,
    ccu.column_name AS foreign_column_name
FROM
    information_schema.table_constraints AS tc
JOIN information_schema.key_column_usage AS kcu
    ON tc.constraint_name = kcu.constraint_name
    AND tc.table_schema = kcu.table_schema
JOIN information_schema.constraint_column_usage AS ccu
    ON ccu.constraint_name = tc.constraint_name
    AND ccu.table_schema = tc.table_schema
WHERE tc.constraint_type = 'FOREIGN KEY' AND tc.table_name=%s;""", (table_name,))
    return self.__cursor.fetchall()

def get_column_types(self, table_name):
    self.__cursor.execute("""SELECT column_name, data_type
                            FROM information_schema.columns
                            WHERE table_schema = 'public' AND table_name = %s
                            ORDER BY table_schema, table_name""", (table_name,))
    return self.__cursor.fetchall()

def get_column_names(self, table_name):
    self.__cursor.execute("""SELECT column_name
                            FROM information_schema.columns
                            WHERE table_schema = 'public' AND table_name = %s
                            ORDER BY table_schema, table_name""", (table_name,))
    return [x[0] for x in self.__cursor.fetchall()]

```

```

def get_table_names(self):

    if self.__table_names is None:
        self.__cursor.execute("""SELECT table_name FROM information_schema.tables
                                WHERE table_schema = 'public'""")
        self.__table_names = [table[0] for table in self.__cursor]

    return self.__table_names

def get_table_data(self, table_name):

    id_column = self.get_column_types(table_name)[0][0]

    cursor = self.__cursor
    try:
        cursor.execute(sql.SQL('SELECT * FROM {} ORDER BY {}
ASC').format(sql.Identifier(table_name), sql.SQL(id_column)))
    except Exception as e:
        return str(e)

    return ( [col.name for col in cursor.description] , cursor.fetchall())

def insert_data(self, table_name, values):
    line = ''
    columns = '('
    for key in values:
        if values[key]:
            line += '%(' + key + ')s,'
            columns += key + ','

    columns = columns[:-1] + ')'

    self.__cursor.execute(
        sql.SQL('INSERT INTO {} {} VALUES (' + line[:-1] + ')')
        .format(sql.Identifier(table_name), sql.SQL(columns)),
        values)
    self.__context.commit()

def generate_data(self, table_name, count):

    types = self.get_column_types(table_name)
    fk_array = self.get_foreign_key_info(table_name)
    select_subquery = ""

```

```

insert_query = "INSERT INTO " + table_name + " ("
for i in range(1,len(types)):
    t = types[i]
    name = t[0]
    type = t[1]
    fk = [x for x in fk_array if x[0] == name]
    if fk:
        select_subquery += '(SELECT {} FROM {} ORDER BY RANDOM(),ser LIMIT
1)'.format(fk[0][2],fk[0][1]))
    elif type == 'integer':
        select_subquery += 'trunc(random()*100)::INT'
    elif type == 'character varying':
        select_subquery += 'chr(trunc(65 + random()*25)::INT) || chr(trunc(65 + ran-
dom()*25)::INT)'
    elif type == 'date':
        select_subquery += "''" date(timestamp '2014-01-10' +
                                random() *
                                (timestamp '2020-01-20' - timestamp '2014-01-10'))""
    elif type == 'time without time zone':
        select_subquery += "time '00:00:00' + DATE_TRUNC('second',RANDOM() * time
'24:00:00')"
    else:
        continue

    insert_query += name
    if i != len(types) - 1:
        select_subquery += ','
        insert_query += ','
    else:
        insert_query += ')'

self.__cursor.execute(insert_query + "SELECT " + select_subquery + "FROM generate_series(1,"
+ str(count) + ") as ser")
self.__context.commit()

def generate_data_for_users(self,size):
    self.generate_data('passport',size)
    self.__cursor.execute("""INSERT INTO users (us-
er_login,user_email,user_reg_date,user_passport_id)
                        SELECT chr(trunc(65 + random()*25)::INT) || chr(trunc(65 + ran-
dom()*25)::INT),
                                chr(trunc(65 + random()*25)::INT) || chr(trunc(65 + ran-
dom()*25)::INT),
                                date(timestamp '2014-01-10' +

```

```

random() * (timestamp '2020-01-20' - timestamp '2014-01-
10')),

s FROM
generate_series((SELECT MAX(passport_id) FROM passport) -
{(SELECT MAX(passport_id) FROM passport)) as s"".format(size-1))
self.__context.commit()

def change_data(self, table_name, values):
    line = ''
    condition = values.pop('condition')
    for key in values:
        if values[key]:
            line += key + '%(' + key + ')s, '

    self.__cursor.execute(
        sql.SQL('UPDATE {} SET '+ line[:-1] + ' WHERE {} ')
        .format(sql.Identifier(table_name), sql.SQL(condition)),
        values)
    self.__context.commit()

def delete_data(self, table_name, cond):
    self.__cursor.execute(
        sql.SQL('DELETE FROM {} WHERE {}'.format(
            sql.Identifier(table_name), sql.SQL(cond)))
    self.__context.commit()

def join_general(self, main_query, condition = ""):
    new_cond = condition
    if condition:
        new_cond = "WHERE " + condition

    t1 = time.time()
    self.__cursor.execute(main_query.format(new_cond))
    t2 = time.time()
    return ((t2 - t1) * 1000, self.__cursor.fetchall())

def join_video_category(self, condition = ""):
    return self.join_general("""SELECT * FROM video as v JOIN category as c ON v.video_category =
c.category_id {} ORDER BY video_id ASC""", condition)

def join_users_passport(self, condition = ""):
    return self.join_general(""" SELECT *
FROM users as u JOIN passport as p ON u.user_passport_id =
p.passport_id {} ORDER BY user_id ASC""", condition)

```

```

def join_comment_video_users(self, condition = ""):
    return self.join_general("""SELECT * FROM commentaries as c
                                JOIN video as v ON c.comment_to_video = v.video_id
                                JOIN users as u ON c.comment_from_user = u.user_id
                                {}
                                ORDER BY comment_id ASC""", condition)

```

Код файла main.py

```

import psycopg2
from db_model import db_model
from controller import controller
import os

def main():
    model = db_model('anton', 'postgres', '821083', '127.0.0.1')
    cont = controller(model)
    view = console_view(model, cont)
    cont.set_view(view)
    func = view.main_menu()
    while func is not None:
        func = func()

if __name__ == '__main__':
    main()

```