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
Панфілов Я.Ю.
Перевірив: Павловський В.І.

Київ – 2020

Ознайомлення з базовими операціями СУБД PostgreSQL

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

Нормалізована модель бази даних

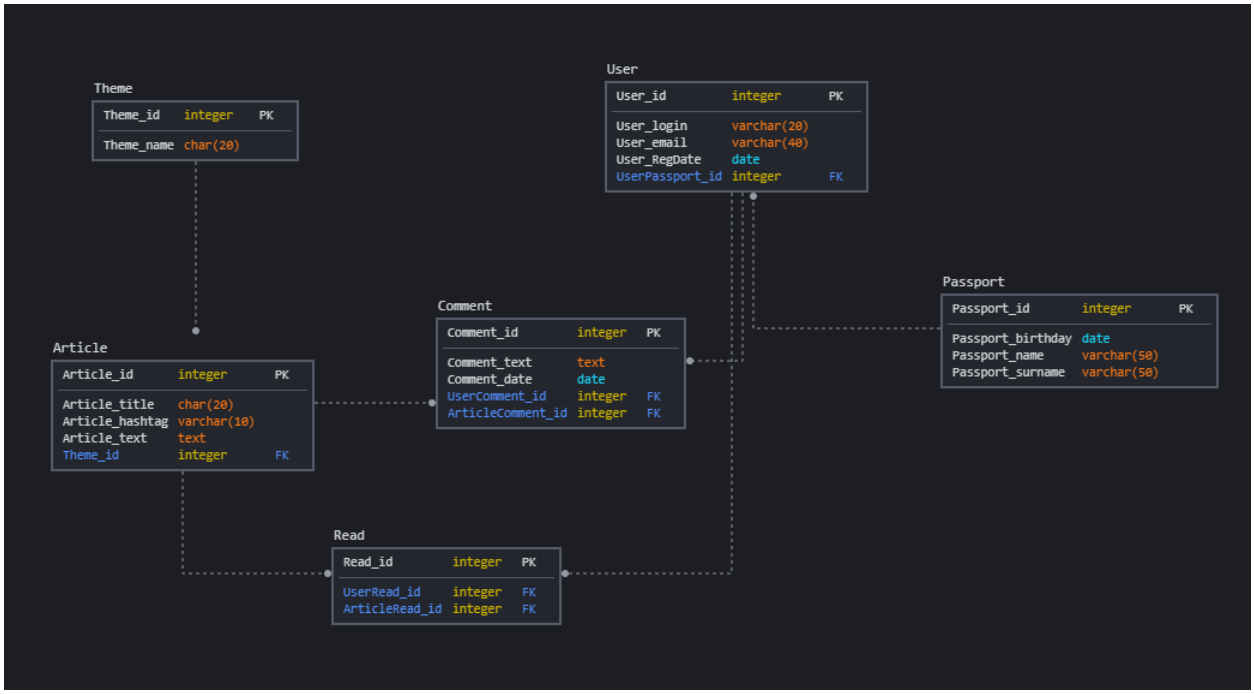


Рис 2.1 – Нормалізована модель даних

Структура програми

Програма створена за патерном MVC (Model-View-Controller). Складається відповідно з модулів Model , View та Controller.

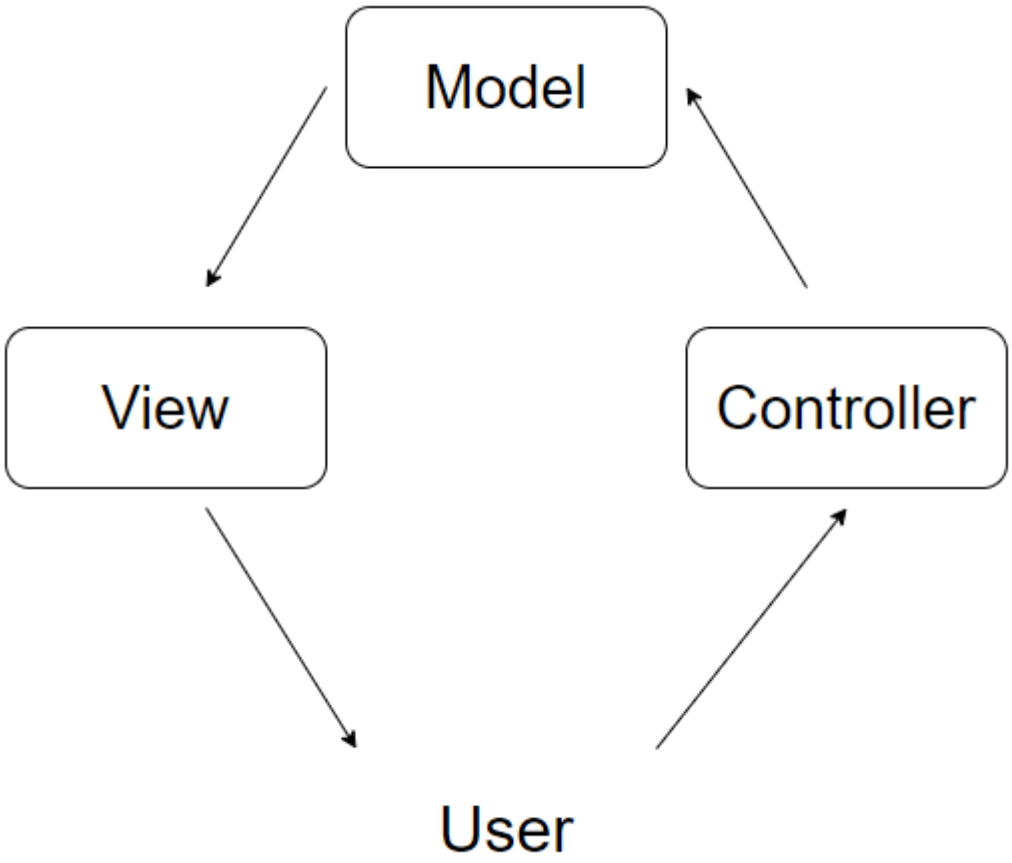


Рис 2.2 – Модель MVC

Опис програми

У класі db_model реалізовані функції , що здійснюють SQL запити до Бази Даних, а також функція, що виконує з’єднання з БД.

У класі console_view реалізовані функції, що використовуються для відображення в консоль пунктів меню, виводу даних з таблиць, тобто функції, що відображують певну інформацію в консоль.

У класі controller реалізовані функції для відповідних меню та допоміжні функції.

Опис структури меню програми

Рис 2.3 – Меню програми

Інтерфейс програми

1. Пункт Show виводить дані про вміст вибраної таблиці.
2. Пункт Insert вставляє введені дані в вибрану таблицю. При введені некоректних даних програма покаже помилку.
3. Пункт Delete дає змогу видалити дані з вибраної таблиці за введеною умовою. При введені неправильної умови програма покаже помилку.
4. Пункт Change дозволяє змінити дані в вибраній таблиці. При введені некоректних даних програма покаже помилку.
5. Пункт Generate дозволяє згенерувати введену кількість полів в вибраній таблиці.
6. Пункт Query проводить пошук в вибраній комбінації таблиці та за певною умовою. При введені неправильних даних програма покаже помилку.
7. Пункт Quit виконує функцію виходу з програми.

Середовище розробки

Середовище для лагодження SQL-запитів до бази даних – PgAdmin4.

Мова програмування – Python 3.7.

Бібліотека взаємодії з PostgreSQL Psycopg2.

Середовище розробки програмного забезпечення – Visual Studio 2019 та Visual Studio Code.

Вибірка елементів з БД

Вибірка елементів з таблиці “article”

```
article
['article_id', 'article_title', 'article_hashtag', 'article_theme', 'article_text']
(1, 'BF', 'WM', 3, 'DQ')
(2, 'DK', 'XH', 6, 'GH')
(3, 'HO', 'FX', 1, 'HV')
(4, 'JH', 'TQ', 4, 'MT')
(5, 'IT', 'XY', 3, 'QG')
```

Рис 2.4 – Елементи таблиці article

Вибірка елементів з таблиці “users”

```
users
['user_id', 'user_login', 'user_email', 'user_reg_date', 'user_passport_id']
(1, 'SX', 'UP', datetime.date(2015, 9, 26), 6)
(2, 'JY', 'MF', datetime.date(2014, 10, 11), 7)
(3, 'AI', 'YI', datetime.date(2018, 6, 21), 8)
(4, 'CJ', 'BA', datetime.date(2017, 11, 15), 9)
(5, 'KV', 'NY', datetime.date(2019, 7, 24), 10)
```

Рис 2.5 – Елементи таблиці users

Метод, який дає змогу отримати дані таблиці

```python
def get_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())
```

Рис 2.6 – Метод для отримання даних з таблиці

Видаленні даних з таблиці

```
DELETE MENU
Choose table or write :r to return
['theme', 'passport', 'users', 'article', 'readby', 'commentaries']
users
users
Columns
['user_id', 'user_login', 'user_email', 'user_reg_date', 'user_passport_id']
WHERE user_id=3
Success
users
['user_id', 'user_login', 'user_email', 'user_reg_date', 'user_passport_id']
(1, 'SX', 'UP', datetime.date(2015, 9, 26), 6)
(2, 'JY', 'MF', datetime.date(2014, 10, 11), 7)
(4, 'CJ', 'BA', datetime.date(2017, 11, 15), 9)
(5, 'KV', 'NY', datetime.date(2019, 7, 24), 10)
```

Рис 2.7 – Приклад видалення з таблиці users

При видаленні даних з однієї таблиці будуть видалені всі пов’язані записи з інших таблиць, що пов’язані з першою.

Метод, який видаляє дані 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 2.8 – Метод видалення даних з таблиці

Додавання даних до таблиці

```
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()
```

Рис 2.9 – Метод додавання даних до таблиці

```
INSERT MENU
choose table or write :r to return
['theme', 'passport', 'users', 'article', 'readby', 'commentaries']
users
Print data to insert into users
user_login ( character varying ): ubri
user_email ( character varying ): ubri11@gmail.com
user_reg_date ( date ): 12-12-2012
user_passport_id ( integer ): 4
Success
users
['user_id', 'user_login', 'user_email', 'user_reg_date', 'user_passport_id']
(1, 'SX', 'UP', datetime.date(2015, 9, 26), 6)
(2, 'JY', 'MF', datetime.date(2014, 10, 11), 7)
(4, 'CJ', 'BA', datetime.date(2017, 11, 15), 9)
(5, 'KV', 'NY', datetime.date(2019, 7, 24), 10)
(6, 'ubri', 'ubri11@gmail.com', datetime.date(2012, 12, 12), 4)
```

Рис 2.10 – Результат додавання запису

Зміна запису 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 2.11 – Метод для зміни даних в таблиці

```
user_login ( character varying ): dzer
user_email ( character varying ): dzer4@gmail.com
user_reg_date ( date ): 01-01-2001
user_passport_id ( integer ): 3
WHERE user_id=2
Success
users
['user_id', 'user_login', 'user_email', 'user_reg_date', 'user_passport_id']
(1, 'SX', 'UP', datetime.date(2015, 9, 26), 6)
(2, 'dzer', 'dzer4@gmail.com', datetime.date(2001, 1, 1), 3)
(4, 'CJ', 'BA', datetime.date(2017, 11, 15), 9)
(5, 'KV', 'NY', datetime.date(2019, 7, 24), 10)
(6, 'ubri', 'ubri11@gmail.com', datetime.date(2012, 12, 12), 4)
```

Рис 2.12 – Результат зміни даних для таблиці users

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_article_theme(self,condition = ""):
    return self.join_general("""SELECT * FROM article as a JOIN theme as t ON a.article_theme = t.theme_id {} ORDER BY article_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_readby_user_article(self,condition = ""):
    return self.join_general("""SELECT *
                            FROM readby as c
                            JOIN article as a ON c.article_read_id = a.article_id
                            JOIN users as u ON c.user_read_id = u.user_id
                            {}
                            ORDER BY read_id""",condition)

def join_comment_user_article(self,condition = ""):
    return self.join_general("""SELECT *
                            FROM commentaries as c
                            JOIN article as a ON c.comment_to_article = a.article_id
                            JOIN users as u ON c.comment_from_user = u.user_id
                            {}
                            ORDER BY comment_id""",condition)
```

Рис 2.13 – Методи для пошуку даних в декількох таблицях

```
Choose query
1 : article + theme
2 : users + passport
3 : comment + user + article
4 : read + user + article
4
QUERY
WHERE user_id=5
execution time 1.004934310913086 ms
(2, 5, 2, 2, 'DK', 'XH', 6, 'GH', 5, 'KV', 'NY', datetime.date(2019, 7, 24), 10)
(3, 5, 4, 4, 'JH', 'TQ', 4, 'MT', 5, 'KV', 'NY', datetime.date(2019, 7, 24), 10)
(4, 5, 1, 1, 'BF', 'WM', 3, 'DQ', 5, 'KV', 'NY', datetime.date(2019, 7, 24), 10)
```

Рис 2.14 – Результат пошуку для таблиць readby, users, article

Додавання N рядків до таблиці

```
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' or type == 'text':
            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'))"""
        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 2.15 – Метод для генерування даних до таблиці

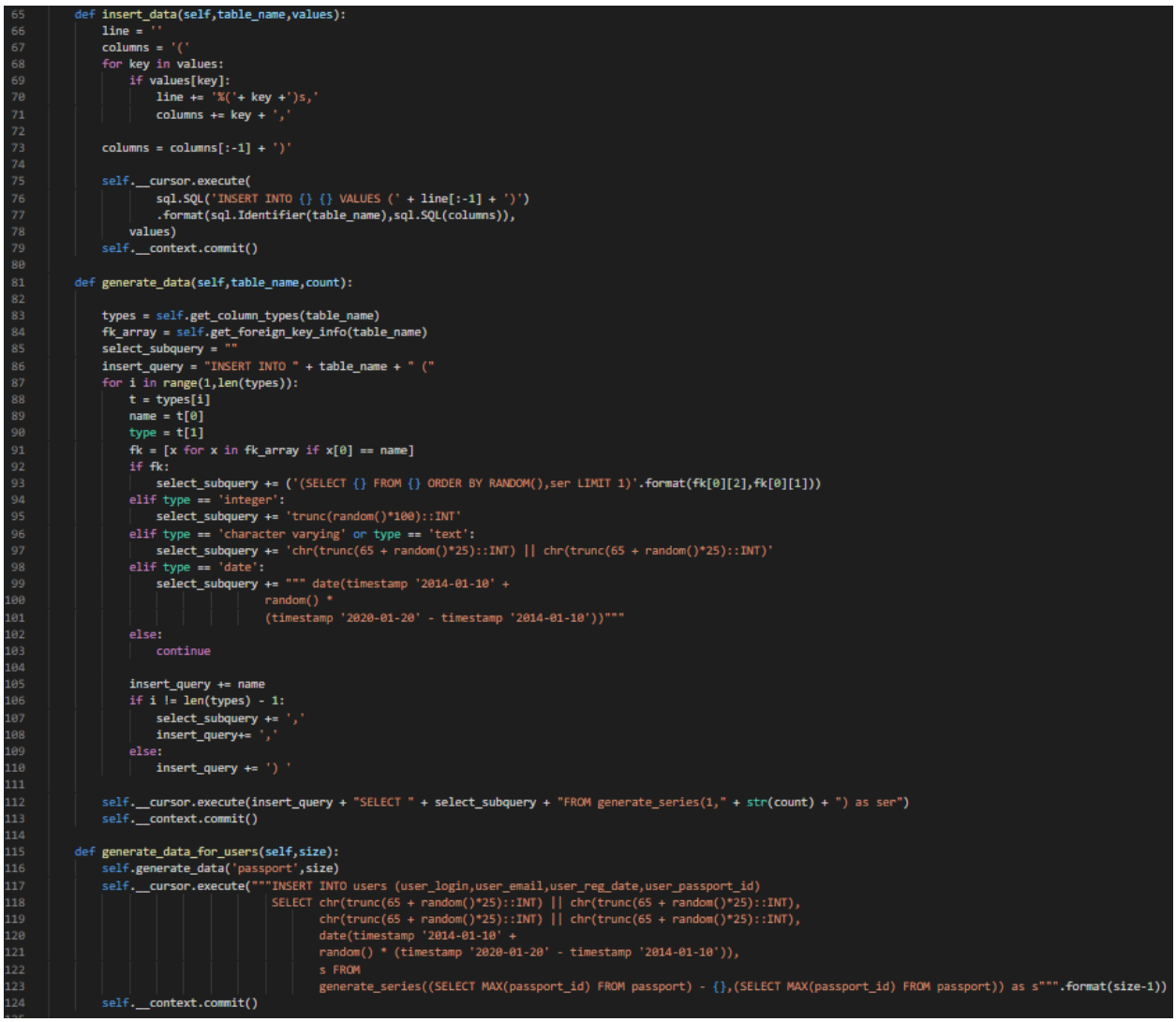
```
GENERATE MENU
Choose table or write :r to return
['theme', 'passport', 'users', 'article', 'readby', 'commentaries']
commentaries
Input N:
5
Success
        (6, 'CN', datetime.date(2016, 11, 11), 10, 4)
        (7, 'RV', datetime.date(2016, 10, 3), 4, 2)
        (8, 'YD', datetime.date(2016, 11, 19), 9, 4)
        (9, 'CA', datetime.date(2017, 1, 29), 6, 6)
        (10, 'TX', datetime.date(2018, 10, 19), 7, 4)
```

Рис 2.16 – Результат додавання 5 рядків до таблиці commentaries

Лістинг модуля “Model”

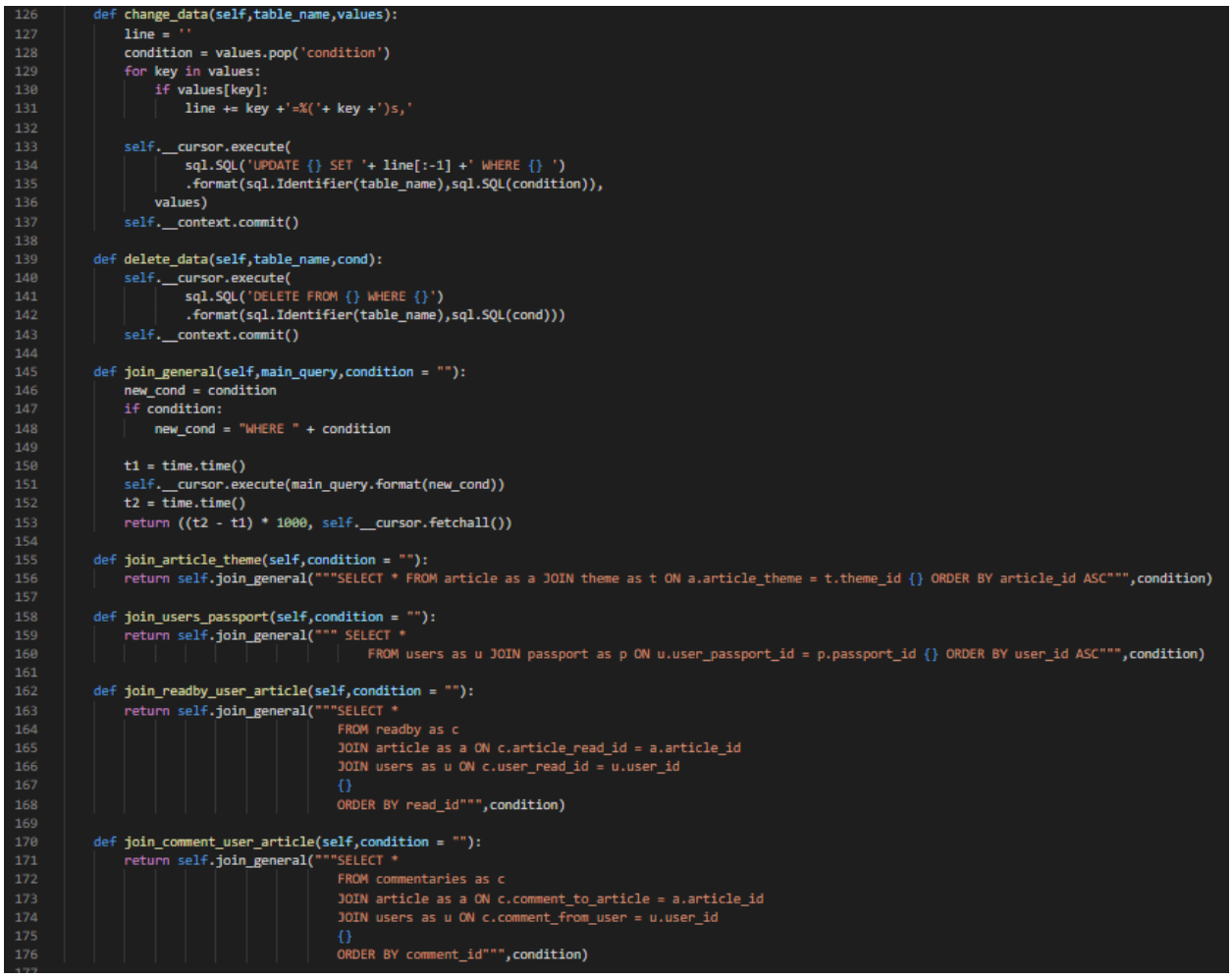
```python
import psycopg2
from psycopg2 import sql
import time

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_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_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())
```

Рис 2.17 – Модуль “Model”



```
    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' or type == 'text':
                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'))"""
            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 (user_login,user_email,user_reg_date,user_passport_id)
                                 SELECT chr(trunc(65 + random()*25)::INT) || chr(trunc(65 + random()*25)::INT),
                                        chr(trunc(65 + random()*25)::INT) || chr(trunc(65 + random()*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()
```

Рис 2.18 – Модуль “Model”



```python
    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_article_theme(self,condition = ""):
        return self.join_general("""SELECT * FROM article as a JOIN theme as t ON a.article_theme = t.theme_id {} ORDER BY article_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_readby_user_article(self,condition = ""):
        return self.join_general("""SELECT *
                                FROM readby as c
                                JOIN article as a ON c.article_read_id = a.article_id
                                JOIN users as u ON c.user_read_id = u.user_id
                                {}
                                ORDER BY read_id""",condition)

    def join_comment_user_article(self,condition = ""):
        return self.join_general("""SELECT *
                                FROM commentaries as c
                                JOIN article as a ON c.comment_to_article = a.article_id
                                JOIN users as u ON c.comment_from_user = u.user_id
                                {}
                                ORDER BY comment_id""",condition)
```

Рис 2.19 – Модуль “Model”

Видалення записів з батьківської таблиці

При видаленні запису з батьківської таблиці автоматично будуть видалені записи з дочірньої таблиці, які відповідали видаленим даним батьківської таблиці.

```
users
['user_id', 'user_login', 'user_email', 'user_reg_date', 'user_passport_id']
(1, 'SX', 'UP', datetime.date(2015, 9, 26), 6)
(2, 'dzer', 'dzer4@gmail.com', datetime.date(2001, 1, 1), 3)
(4, 'CJ', 'BA', datetime.date(2017, 11, 15), 9)
(5, 'KV', 'NY', datetime.date(2019, 7, 24), 10)
(6, 'ubri', 'ubri11@gmail.com', datetime.date(2012, 12, 12), 4)
```

Рис 2.20 – Дочірня таблиця до видалення запису

```
passport
Columns
['passport_id', 'passport_name', 'passport_surname', 'passport_birthday']
WHERE passport_id=3
Success
```

Рис 2.21 – Видалення запису з батьківської таблиці

```
sers
'user_id', 'user_login', 'user_email', 'user_reg_date', 'user_passport_id']
1, 'SX', 'UP', datetime.date(2015, 9, 26), 6)
4, 'CJ', 'BA', datetime.date(2017, 11, 15), 9)
5, 'KV', 'NY', datetime.date(2019, 7, 24), 10)
6, 'ubri', 'ubri11@gmail.com', datetime.date(2012, 12, 12), 4)
```

Рис 2.22 – Дочірня таблиця після видалення запису

Операція вставки запису в дочірню таблицю

```
choose table or write :r to return
['theme', 'passport', 'users', 'article', 'readby', 'commentaries']
article
Input data article
article_title ( character varying ): title
article_hashtag ( character varying ): frs
article_theme ( integer ): 15
article_text ( text ): ksks
ОШИБКА:  INSERT или UPDATE в таблице "article" нарушает ограничение внешнего ключа "article_theme_idfk"
DETAIL:  Ключ (article_theme)=(15) отсутствует в таблице "theme".

INSERT MENU
choose table or write :r to return
['theme', 'passport', 'users', 'article', 'readby', 'commentaries']
```

Рис 2.23 – Помилка при вставці запису в дочірню таблицю

При спробі вставці запису в дочірню таблицю отримано повідомлення про помилку, оскільки в батьківській таблиці відсутні дані до яких користувач хоче додати запис. Після повідомлення помилки видано меню для повторного введення даних.

```
article
Input data article
article_title ( character varying ): article
article_hashtag ( character varying ): frs
article_theme ( integer ): 3
article_text ( text ): ksks
Success
```

Рис 2.24 – Спроба вставки даних в дочірню таблицю

```
article
['article_id', 'article_title', 'article_hashtag', 'article_theme', 'article_text']
(2, 'DK', 'XH', 6, 'GH')
(3, 'HO', 'FX', 1, 'HV')
(4, 'JH', 'TQ', 4, 'MT')
(5, 'IT', 'XY', 3, 'QG')
(6, 'TS', 'RV', 6, 'BF')
(7, 'HJ', 'FV', 6, 'XL')
(8, 'NK', 'RQ', 4, 'QF')
(9, 'HJ', 'WL', 4, 'SV')
(10, 'QK', 'CK', 6, 'DS')
(13, 'article', 'frs', 3, 'ksks')
```

Рис 2.25 – Результат вдалої вставки в дочірню таблицю

Модуль “Controller”

```python
class controller():
    def __init__(self, model):
        self.__model = model
        self.__query_data = ["article + theme","users + passport","comment + user + article", "read + user + article"]

    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 == ':menu':
            return self.get_view_func("main_menu")

        if data == ':r':
            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 == '7':
            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_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
```

Рис 2.26 – Модуль “controller”

```
    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 = [x[0] for x in self.__model.get_column_types(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)
            res_func = self.get_view_func("cond_query_menu",int_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
```

Рис 2.27 – Модуль “controller”

```python
    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 = [x[0] for x in self.__model.get_column_types(table)]

        return self.get_view_func("print_message",message,self.get_view_func("delete_row_menu",table,column_arr))
```

Рис 2.28 – Модуль “controller”

```
183     def generate_data(self,table_name,data):
184         res = self.check_input(data,"choose_generate_menu")
185 
186         if res is not None:
187             return res
188 
189         message = " "
190         data_int = 0
191 
192         try:
193             data_int = int(data)
194         except Exception as e:
195             return self.get_view_func("print_message",str(e),self.get_view_func("generate_size_menu",table_name))
196 
197 
198         if res is not None:
199             return res
200 
201         if table_name == 'users':
202             try:
203                 self.__model.generate_data_for_users(data_int)
204                 message = "Success"
205             except Exception as e:
206                 message = str(e)
207                 self.__model.clear_transaction()
208         else:
209             try:
210                 self.__model.generate_data(table_name,data_int)
211                 message = "Success"
212             except Exception as e:
213                 message = str(e)
214                 self.__model.clear_transaction()
215 
216         return self.get_view_func("print_message",message,self.get_view_func("generate_size_menu",table_name))
217 
218     def cond_query_menu(self,query_num,cond):
219 
220         if cond == ':menu':
221             return self.get_view_func("main_menu")
222 
223         if cond == ':r':
224             return self.get_view_func("choose_query_menu",self.__query_data)
225 
226         if cond == ':q':
227             return self.get_view_func("quit")
228 
229         query_func = None
230         if query_num == 1:
231             query_func = getattr(self.__model, "join_article_theme")
232         elif query_num == 2:
233             query_func = getattr(self.__model, "join_users_passport")
234         elif query_num == 3:
235             query_func = getattr(self.__model, "join_comment_user_article")
236         elif query_num == 4:
237             query_func = getattr(self.__model, "join_readby_user_article")
238 
239         ret_func = None
240         try:
241             data = query_func(cond)
242             ret_func = self.get_view_func("print_table",("execution time {} ms".format(data[0]),data[1]),self.get_view_func("cond_query_menu",query_num))
243         except Exception as e:
244             ret_func = self.get_view_func("print_message",str(e),self.get_view_func("cond_query_menu",query_num))
245             self.__model.clear_transaction()
246 
247         return ret_func
```

Рис 2.29 – Модуль “controller”

Модуль “Console_view”

```python
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):
        print(table[0])
        for row in table[1]:
            print(row)
        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('Select:')
        print('1)Show')
        print('2)Insert')
        print('3)Delete')
        print('4)Change')
        print('5)Generate')
        print('6)Querry')
        print('7)Quit')
        char = chr(msvcrt.getch()[0])
        return self.__controller.main_menu(char)

    def show_menu(self):
        table_names = self.__model.get_table_names()
        print('SHOW MENU')
        print('Choose table or write :r to return')
        print(table_names)
        answer = input()
        return self.__controller.show_menu(answer)

    def choose_delete_menu(self):
        table_names = self.__model.get_table_names()
        print('DELETE MENU')
        print('Choose table or write :r to return')
        print(table_names)
        answer = input()
        return self.__controller.choose_delete_menu(answer)

    def choose_insert_menu(self):
        table_names = self.__model.get_table_names()
        print('INSERT MENU')
        print('choose table or write :r to return')
        print(table_names)
        answer = input()
        return self.__controller.choose_insert_menu(answer)
```

Рис 2.30 – Модуль “console_view”

```
    def choose_change_menu(self):
        table_names = self.__model.get_table_names()
        print('CHANGE MENU')
        print('Choose table or write :r to return')
        print(table_names)
        answer = input()
        return self.__controller.choose_change_menu(answer)

    def choose_generate_menu(self):
        table_names = self.__model.get_table_names()
        print('GENERATE MENU')
        print('Choose table or write :r to return')
        print(table_names)
        answer = input()
        return self.__controller.choose_generate_menu(answer)

    def choose_query_menu(self,query_list):
        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('Input data',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('Input data',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(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('Input N:')
        answer = input()
        return self.__controller.generate_data(table_name,answer)

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

Рис 2.31 – Модуль “console_view”