НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
"КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ
СІКОРСЬКОГО"

Кафедра програмного забезпечення комп'ютерних систем

Лабораторна робота №2
із дисципліни «Бази даних»
на тему **«Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL»**

Виконав:
студент 3 курсу ФПМ групи КП-82
Новохацький Владислав Андрійович

Прийняв: Радченко К.О.

	Бали
Якість виконання	
Термін здачі	
Сумарний бал	

КИЇВ — 2020

Мета роботи

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

Постановка завдання

1. Реалізувати функції внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі No1, 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 програми

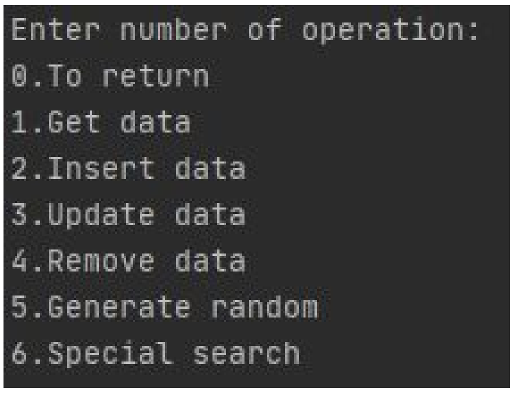
Програма допомагає взаємодіяти з базою даних завдяки консольному інтерфейсу, є можливість додавати, змінювати видаляти елементи бази даних.

Видалення елементів відбувається каскадно, тобто видаляються всі елементи, що втрачають своє значення в базі даних (реалізовано програмно)

Програма побудована на згідно патерну MVC, весь код розділений на три складові: model, view, controller, код розподілений по окремим файлам.

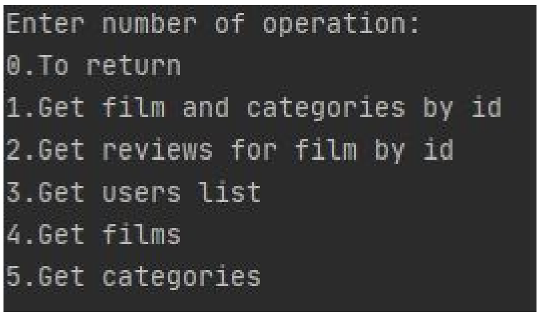
При випадковому генеруванні даних також генеруються випадки між різними таблицями.

Результати виконання роботи



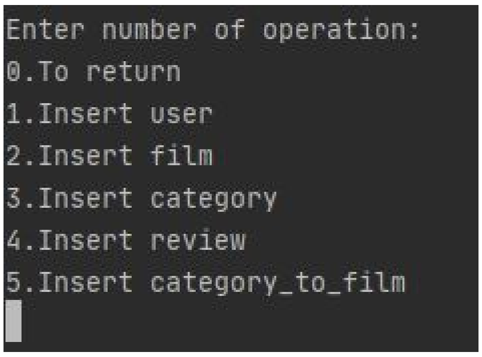
```
Enter number of operation:
0.To return
1.Get data
2.Insert data
3.Update data
4.Remove data
5.Generate random
6.Special search
```

Рис. 1. Основне меню програми



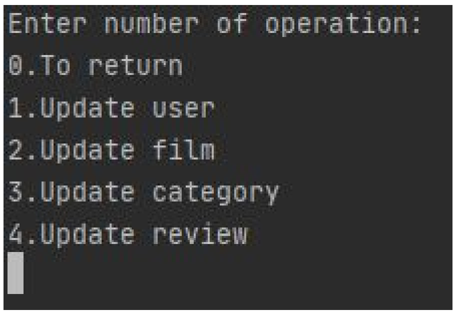
```
Enter number of operation:
0.To return
1.Get film and categories by id
2.Get reviews for film by id
3.Get users list
4.Get films
5.Get categories
```

Рис. 2. Меню отримання даних з таблиці



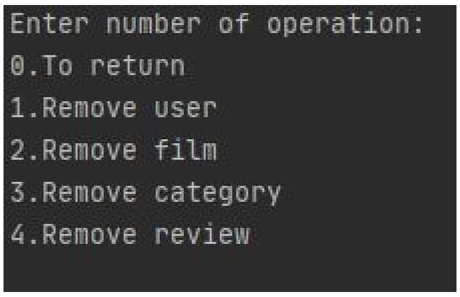
```
Enter number of operation:
0.To return
1.Insert user
2.Insert film
3.Insert category
4.Insert review
5.Insert category_to_film
```

Рис. 3. Меню введення нових даних в таблицю



```
Enter number of operation:
0.To return
1.Update user
2.Update film
3.Update category
4.Update review
```

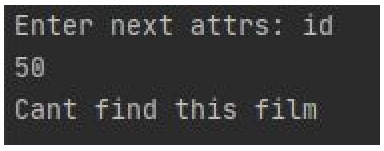
Рис. 4. Меню модифікування даних таблиці



```
Enter number of operation:
0.To return
1.Remove user
2.Remove film
3.Remove category
4.Remove review
```

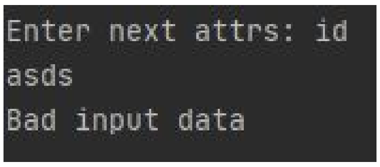
Рис. 5. Меню видалення даних таблиці

Завдання 1



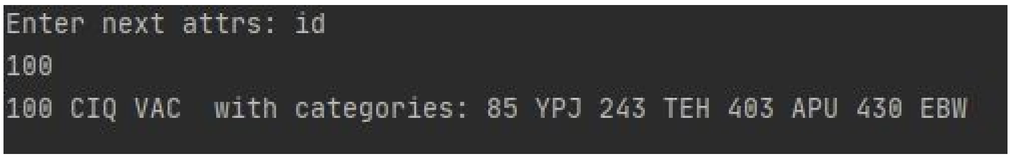
```
Enter next attrs: id
50
Cant find this film
```

Рис. 6. Випадок при пошуку, видаленню, модифікуванню елементу за id, коли його немає у базі



```
Enter next attrs: id
asds
Bad input data
```

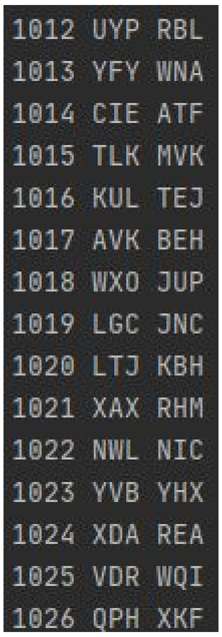
Рис. 7. Випадок при введенні помилкового типу даних



```
Enter next attrs: id
100
100 CIQ VAC  with categories: 85 YPJ 243 TEH 403 APU 430 EBW
```

Рис. 8. Успішне отримання даних з таблиці

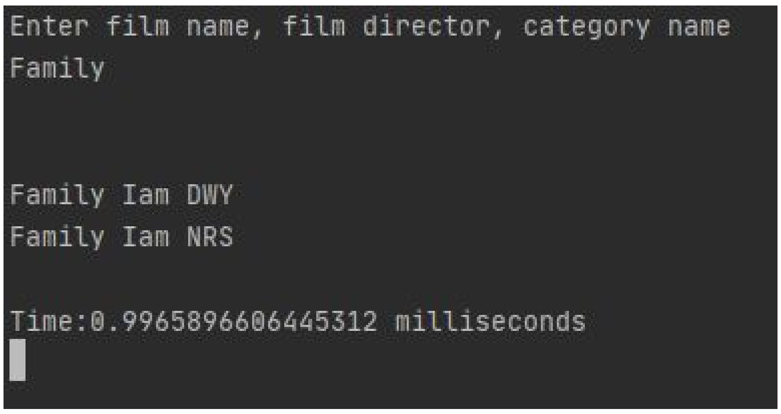
Завдання 2



```
1012 UYP RBL
1013 YFY WNA
1014 CIE ATF
1015 TLK MVK
1016 KUL TEJ
1017 AVK BEH
1018 WXO JUP
1019 LGC JNC
1020 LTJ KBH
1021 XAX RHM
1022 NWL NIC
1023 YVB YHX
1024 XDA REA
1025 VDR WQI
1026 QPH XKF
```

Рис. 9. Фрагмент згенерованих випадковим чином даних

Завдання 3



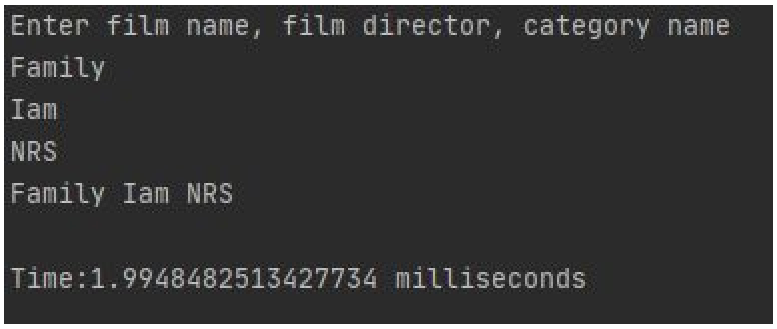
```
Enter film name, film director, category name
Family


Family Iam DWY
Family Iam NRS

Time:0.9965896606445312 milliseconds
```

,

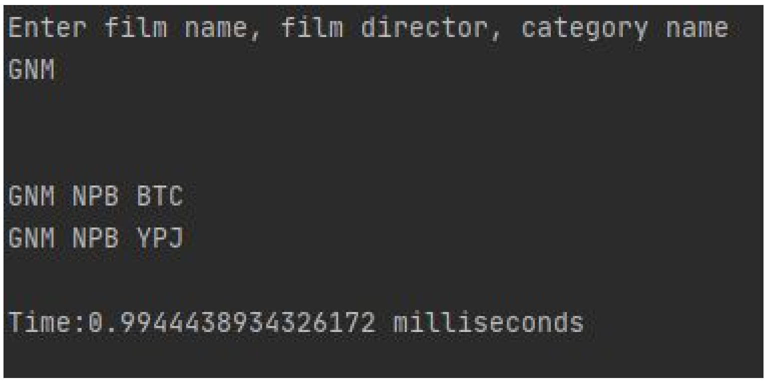
Рис. 10. Пошук за одним з трьох атрибутів та час пошуку



```
Enter film name, film director, category name
Family
Iam
NRS
Family Iam NRS

Time:1.9948482513427734 milliseconds
```

Рис. 10. Пошук за трьома атрибутами та час пошуку



```
Enter film name, film director, category name
GNM


GNM NPB BTC
GNM NPB YPJ

Time:0.9944438934326172 milliseconds
```

Рис. 11. Пошук з іншими вхідними даними

Завдання 4

main.py

```
import controller
import view
import model

c = controller.Controller(model.Model("dbname=reviews
user=postgres password=12345"), view.View())
c.show_start_menu()
```

model.py

```
import backend


class Model(object):

    def __init__(self, db_string):
        self.db_string = db_string
        backend.set_data_base(db_string)

    def __del__(self):
        backend.close_database()

    def add_category(self, name):
        return backend.add_category(name)
    def add_user(self, username, password, ava_url):
        return backend.add_user(username, password, ava_url)

    def add_film(self, name, director):
        return backend.add_film(name, director)

    def add_review(self, comment, rating, user_id, film_id):
        return backend.add_review(comment, rating, user_id,
film_id)

    def add_film_category(self, film_id, category_id):
        return backend.add_film_category(film_id, category_id)

    def edit_category(self, item_id, name):
        return backend.edit_category(item_id, name)

    def edit_user(self, item_id, username, password, ava_url):
        return backend.edit_user(item_id, username, password,
ava_url)

    def edit_film(self, item_id, name, director):
        return backend.edit_film(item_id, name, director)

    def edit_review(self, item_id, comment, rating, user_id):
        return backend.edit_review(item_id, comment, rating,
user_id)

    def remove_category(self, item_id):
        return backend.remove_category(item_id)

    def remove_user(self, item_id):
        return backend.remove_user(item_id)

    def remove_film(self, item_id):
        return backend.remove_film(item_id)

    def remove_review(self, item_id):
        return backend.remove_review(item_id)

    def remove_film_category(self, film_id, category_id):
        return backend.remove_film_category(film_id, category_id)

    def get_film_and_categories(self, film_id):
        return backend.get_film_and_categories(film_id)

    def get_reviews_to_film(self, film_id):
        return backend.get_reviews_to_film(film_id)

    def get_users(self):
        return backend.get_users()

    def get_films(self):
        return backend.get_films()
    def get_categories(self):
        return backend.get_categories()

    def special_find(self, name, director, category_name):
        return backend.special_find(name, director,
category_name)

    def generate_random(self):
        return backend.generate_random()
```

backend.py

```
import psycopg2
import time

# dbname=reviews user=postgres password=12345

con = psycopg2.connect("dbname=reviews user=postgres
password=12345")
cur = con.cursor()


def set_data_base(string):
   global con
   global cur
   con = psycopg2.connect(string)
   cur = con.cursor()


def close_database():
   cur.close()
   con.close()


def add_category(name):
   try:
       cur.execute("INSERT INTO categories (name) VALUES
('{0}')".format(name))
       con.commit()
       return "Completed"
   except psycopg2.ProgrammingError:
       con.rollback()
       return "Bad input data"
   except Exception as error:
       con.rollback()
       return str(error)


def add_user(username, password, ava_url):
   try:
       cur.execute("INSERT INTO users (username,password,ava_url)
VALUES ('{0}','{1}','{2}') RETURNING id"
                  .format(username, password, ava_url))
       con.commit()
```

```
        return "Completed"
    except psycopg2.ProgrammingError:
        con.rollback()
        return "Bad input data"
    except Exception as error:
        con.rollback()
        return str(error)


def add_film(name, director):
    try:
        cur.execute("INSERT INTO films (name, director) VALUES
('{0}','{1}')"
                    .format(name, director))
        con.commit()
        return 'Completed'
    except psycopg2.ProgrammingError:
        con.rollback()
        return "Bad input data"
    except Exception as error:
        con.rollback()
        return str(error)


def add_film_category(film_id, category_id):
    try:
        cur.execute("SELECT 0 id from films WHERE
id=({0})".format(film_id))
        is_exits = cur.fetchone()[0]
        if is_exits != 0:
            return "Can't find this film"
        cur.execute("SELECT 0 id from categories WHERE
id=({0})".format(category_id))
        is_exits = cur.fetchone()[0]
        if is_exits != 0:
            return "Can't find this category"
        cur.execute("INSERT INTO film_category (film_id,
category_id) VALUES ({0}, {1})"
                    .format(film_id, category_id))
        con.commit()
        return 'Completed'
    except psycopg2.ProgrammingError:
        con.rollback()
        return "Bad input data"
    except Exception as error:
        con.rollback()
        return str(error)


def add_review(comment, rating, user_id, film_id):
    try:
        cur.execute("SELECT 0 id from users WHERE
id=({0})".format(user_id))
        if cur.fetchone() is None:
            return "Can't find this user"
        cur.execute("SELECT 0 id from films WHERE
id=({0})".format(film_id))
        is_exits = cur.fetchone()[0]
        if is_exits != 0:
            return "Can't find this film"
        cur.execute("INSERT INTO reviews (comment, rating,
user_id) VALUES ('{0}',{1},{2}) RETURNING id"
                    .format(comment, rating, user_id))
        last_id = cur.fetchone()[0]
        cur.execute("INSERT INTO film_review (film_id, review_id)
VALUES ({0}, {1})"
                    .format(film_id, last_id))
        con.commit()
        return 'Completed'
    except psycopg2.ProgrammingError:
        con.rollback()
        return "Bad input data"
    except Exception as error:
        con.rollback()
        return str(error)


def edit_category(item_id, name):
    try:
        cur.execute("SELECT 0 id from categories WHERE
id=({0})".format(item_id))
        if cur.fetchone() is None:
            return "Can't find this category"
        cur.execute("UPDATE categories SET name='{0}' WHERE
id={1}".format(name, item_id))
        con.commit()
        return "Completed"
    except psycopg2.ProgrammingError:
        con.rollback()
        return "Bad input data"
    except Exception as error:
        con.rollback()
        return str(error)


def edit_user(item_id, username, password, ava_url):
    try:
        cur.execute("SELECT 0 id from users WHERE
id=({0})".format(item_id))
        if cur.fetchone() is None:
            return "Can't find this user"
        cur.execute("""UPDATE users SET username='{0}',
password='{1}', ava_url='{2}')
                    VALUES ('{0}','{1}','{2}') WHERE id={3}"""
                    .format(username, password, ava_url, item_id))
        con.commit()
        return "Completed"
    except psycopg2.ProgrammingError:
        con.rollback()
        return "Bad input data"
```

```
   except Exception as error:
       con.rollback()
       return str(error)


def edit_film(item_id, name, director):
   try:
       cur.execute("SELECT 0 id from films WHERE
id=({0})".format(item_id))
       if cur.fetchone() is None:
           return "Can't find this film"
       cur.execute("UPDATE films SET name='{0}', director='{1}'
WHERE id={2}"
                   .format(name, director, item_id))
       con.commit()
       return 'Completed'
   except psycopg2.ProgrammingError:
       con.rollback()
       return "Bad input data"
   except Exception as error:
       con.rollback()
       return str(error)


def edit_review(item_id, comment, rating, user_id):
   try:
       cur.execute("SELECT 0 id from reviews WHERE
id=({0})".format(item_id))
       if cur.fetchone() is None:
           return "Can't find this review"
       cur.execute("SELECT 0 id from users WHERE
id=({0})".format(user_id))
       if cur.fetchone() is None:
           return "Can't find this user"
       cur.execute("UPDATE reviews SET comment='{0}', rating={1},
user_id={2} WHERE id={3}"
                   .format(comment, rating, user_id, item_id))
       con.commit()
       return 'Completed'
   except psycopg2.ProgrammingError:
       con.rollback()
       return "Bad input data"
   except Exception as error:
       con.rollback()
       return str(error)


def remove_category(item_id):
   try:
       cur.execute("SELECT 0 id from categories WHERE
id=({0})".format(item_id))
       if cur.fetchone() is None:
           return "Can't find this category"
       cur.execute("DELETE FROM film_category WHERE
category_id={0}".format(item_id))
```

```
       cur.execute("DELETE FROM categories WHERE
id={0}".format(item_id))
       con.commit()
       return "Completed"
   except psycopg2.ProgrammingError:
       con.rollback()
       return "Bad input data"
   except Exception as error:
       con.rollback()
       return str(error)


def remove_user(item_id):
   try:
       cur.execute("SELECT 0 id from users WHERE
id=({0})".format(item_id))
       if cur.fetchone() is None:
           return "Can't find this user"
       cur.execute("UPDATE reviews SET user_id=NULL WHERE
user_id={0}".format(item_id))
       cur.execute("DELETE FROM users WHERE
id={0}".format(item_id))
       con.commit()
       return "Completed"
   except psycopg2.ProgrammingError:
       con.rollback()
       return "Bad input data"
   except Exception as error:
       con.rollback()
       return str(error)


def remove_film(item_id):
   try:
       cur.execute("SELECT 0 id from films WHERE
id=({0})".format(item_id))
       if cur.fetchone() is None:
           return "Can't find this film"
       cur.execute("DELETE FROM film_category WHERE
film_id={0}".format(item_id))
       cur.execute("SELECT id FROM film_review WHERE
film_id={0}".format(item_id))
       review_ids = cur.fetchall()
       cur.execute("DELETE FROM film_review WHERE
film_id={0}".format(i1tem_id))
       for item in review_ids:
           cur.execute("DELETE FROM reviews WHERE
id={0}".format(item[0]));
       cur.execute("DELETE FROM films WHERE
id={0}".format(item_id))
       con.commit()
       return "Completed"
   except Exception as error:
       con.rollback()
       return str(error)
```

```
def remove_review(item_id):
   try:
       cur.execute("SELECT 0 id from reviews WHERE
id=({0})".format(item_id))
       if cur.fetchone() is None:
           return "Can't find this review"
       cur.execute("DELETE FROM film_review WHERE
review_id={0}".format(item_id))
       cur.execute("DELETE FROM reviews WHERE
id={0}".format(item_id))
       con.commit()
       return "Completed"
   except psycopg2.ProgrammingError:
       con.rollback()
       return "Bad input data"
   except Exception as error:
       con.rollback()
       return str(error)


def remove_film_category(film_id, category_id):
   try:
       cur.execute("SELECT 0 id from films WHERE
id=({0})".format(film_id))
       if cur.fetchone() is None:
           return "Can't find this film"
       cur.execute("SELECT 0 id from categories WHERE
id=({0})".format(category_id))
       is_exits = cur.fetchone()[0]
       if is_exits != 0:
           return "Can't find this category"
       cur.execute("DELETE FROM film_category WHERE film_id={0}
AND category_id={1}"
                   .format(film_id, category_id))
       con.commit()
       return 'Completed'
   except psycopg2.ProgrammingError:
       con.rollback()
       return "Bad input data"
   except Exception as error:
       con.rollback()
       return str(error)


def get_film_and_categories(film_id):
   try:
       string = ''
       cur.execute("SELECT * from films WHERE
id={0}".format(film_id))
       item = cur.fetchone()
       if item is None:
           return "Cant find this film"
       for i in item:
           string += str(i) + " "
       cur.execute('''SELECT categories.id, categories.name from
films JOIN film_category ON films.id=film_category.film_id
               JOIN categories ON
film_category.category_id=categories.id WHERE
films.id={0}'''.format(film_id))
       items = cur.fetchall()
       string += " with categories: "
       for item in items:
           for i in item:
               string += str(i) + ' '
       return string
   except psycopg2.ProgrammingError:
       con.rollback()
       return "Bad input data"
   except Exception as error:
       con.rollback()
       return str(error)


def get_reviews_to_film(film_id):
   try:
       cur.execute("SELECT * from films WHERE
id={0}".format(film_id))
       item = cur.fetchone()
       if item is None:
           return "Cant find this film"
       string = "" + str(film_id) + ": \n"
       cur.execute('''SELECT
reviews.id,reviews.rating,reviews.user_id,reviews.comment
               FROM reviews LEFT JOIN film_review ON
film_review.review_id=reviews.id
               WHERE film_review.film_id =
{0}'''.format(film_id))
       items = cur.fetchall()
       for item in items:
           for i in item:
               string += str(i) + ' '
           string += '\n'
       return string
   except psycopg2.ProgrammingError:
       con.rollback()
       return "Bad input data"
   except Exception as error:
       con.rollback()
       return str(error)


def get_users():
   try:
       string = "users: \n"
       cur.execute("SELECT * FROM users".format())
       items = cur.fetchall()
       if items is None:
           return "Cant find any user"
       for item in items:
           for i in item:
               string += str(i) + ' '
           string += '\n'
       return string
   except psycopg2.ProgrammingError:
       con.rollback()
       return "Bad input data"
   except Exception as error:
       con.rollback()
       return str(error)


def get_films():
   try:
       string = "films: \n"
       cur.execute("SELECT * FROM films".format())
       items = cur.fetchall()
       if items is None:
           return "Cant find any user"
       for item in items:
           for i in item:
               string += str(i) + ' '
           string += '\n'
       return string
   except psycopg2.ProgrammingError:
       con.rollback()
       return "Bad input data"
   except Exception as error:
       con.rollback()
       return str(error)


def get_categories():
   try:
       string = "categories: \n"
       cur.execute("SELECT * FROM categories".format())
       items = cur.fetchall()
       if items is None:
           return "Cant find any category"
       for item in items:
           for i in item:
               string += str(i) + ' '
           string += '\n'
       return string
   except psycopg2.ProgrammingError:
       con.rollback()
       return "Bad input data"
   except Exception as error:
       con.rollback()
       return str(error)


def special_find(name, director, category_name):
   try:
       start_time = time.time()
       query = '''
               SELECT films.name, films.director, categories.name
               FROM films
               JOIN film_category ON
films.id=film_category.film_id
               JOIN categories ON
categories.id=film_category.category_id'''
       if name:
           query += " WHERE films.name LIKE '{0}'".format(name)
       if director and not name:
           query += " WHERE director LIKE '{0}'
".format(director)
       elif director:
           query += " AND director LIKE '{0}'".format(director)
       if category_name and not name and not director:
           query += " WHERE categories.name LIKE
'{0}'".format(category_name)
       elif category_name:
           query += " AND categories.name LIKE
'{0}'".format(category_name)
       cur.execute(query)
       string = ''
       items = cur.fetchall()
       if not items:
           return "no items"
       for item in items:
           for i in item:
               string += str(i) + ' '
           string += '\n'
       string += "\nTime:" + str((time.time() - start_time)*1000)
+ " milliseconds"
       return string
   except Exception as error:
       return str(error)


def generate_random():
   try:
       cur.execute('''
               INSERT INTO films (name,director)
               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) ||
chr(trunc(65+random()*25)::int) ||
chr(trunc(65+random()*25)::int)
               FROM generate_series(1,100)''')
       cur.execute('''
               INSERT INTO categories (name)
               SELECT
               chr(trunc(65+random()*25)::int) ||
chr(trunc(65+random()*25)::int) ||
chr(trunc(65+random()*25)::int)
```

```
               FROM generate_series(1,100)''')
       cur.execute('''
               INSERT INTO users (username,password,ava_url)
               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) ||
chr(trunc(65+random()*25)::int) ||
chr(trunc(65+random()*25)::int),
               chr(trunc(65+random()*25)::int) ||
chr(trunc(65+random()*25)::int) ||
chr(trunc(65+random()*25)::int)
               FROM generate_series(1,100)''')
       cur.execute('''
               DO
               $$
               DECLARE
                   i record;
               BEGIN
               FOR i_num_count IN 1 .. 100 BY 1
                   LOOP
                       INSERT INTO film_category
(film_id,category_id) VALUES (
                       (SELECT id FROM films
                       ORDER BY random()
                       LIMIT 1),
                       (SELECT id FROM categories
                       ORDER BY random()
                       LIMIT 1)
                       );
                   END LOOP;
               END;
               $$
               ;''')
       con.commit()
       return "Complete"
   except:
       con.rollback()
       return "Failed"
```

view.py

```
import os


class View(object):

    @staticmethod
    def show_start_menu():
        print('''Enter number of operation:
0.To return
1.Get data
2.Insert data
3.Update data
4.Remove data
5.Generate random
6.Special search''')

    @staticmethod
    def show_get_menu():
        print('''Enter number of operation:
0.To return
1.Get film and categories by id
2.Get reviews for film by id
3.Get users list
4.Get films
5.Get categories''')

    @staticmethod
    def show_insert_menu():
        print('''Enter number of operation:
0.To return
1.Insert user
2.Insert film
3.Insert category
4.Insert review
5.Insert category_to_film''')

    @staticmethod
    def show_update_menu():
        print('''Enter number of operation:
0.To return
1.Update user
2.Update film
3.Update category
4.Update review''')

    @staticmethod
    def show_remove_menu():
        print('''Enter number of operation:
0.To return
1.Remove user
2.Remove film
3.Remove category
4.Remove review''')

    @staticmethod
    def show_special_search():
        print('Enter film name, film director, category name')

    @staticmethod
    def show_input(array_of_attrs):
        string = 'Enter next attrs: '
        for item in array_of_attrs:
            string += item + " "
        print(string)

    @staticmethod
    def clear():
        os.system('cls')
```

controller.py

```
class Controller(object):

    def __init__(self, model, view):
        self.model = model
        self.view = view

    def show_start_menu(self):
        self.view.clear()
        while 1:
            self.view.clear()
            self.view.show_start_menu()
            try:
                operation = int(input())
            except:
                continue
            if operation == 1:
                self.show_get_menu()
            elif operation == 2:
                self.show_insert_menu()
            elif operation == 3:
                self.show_update_menu()
            elif operation == 4:
                self.show_remove_menu()
            elif operation == 5:
                print(self.model.generate_random())
            elif operation == 6:
                self.show_special_search()
            elif operation == 0:
                return
            else:
                self.view.clear()
                continue

    def show_get_menu(self):
        self.view.clear()
        while 1:
            self.view.clear()
            self.view.show_get_menu()
            try:
                operation = int(input())
            except:
                continue
            if operation == 1:
                return self.show_get(1, 1)
            elif operation == 2:
                return self.show_get(2, 1)
            elif operation == 3:
                return self.show_get(3)
            elif operation == 4:
                return self.show_get(4)
            elif operation == 5:
                return self.show_get(5)
            elif operation == 0:
                return
            else:
                self.view.clear()
                continue

    def show_insert_menu(self):
        self.view.clear()
        while 1:
            self.view.clear()
            self.view.show_insert_menu()
            try:
                operation = int(input())
            except:
                continue
            return self.show_insert(operation)

    def show_update_menu(self):
        self.view.clear()
        while 1:
            self.view.clear()
            self.view.show_update_menu()
            try:
                operation = int(input())
            except:
                continue
            return self.show_update(operation)

    def show_remove_menu(self):
        self.view.clear()
        while 1:
            self.view.clear()
            self.view.show_remove_menu()
            try:
                operation = int(input())
            except:
                continue
            return self.show_remove(operation)

    def show_get(self, operation, item_id=-1):
        self.view.clear()
        try:
            while 1:
                self.view.clear()
                if item_id != -1:
                    self.view.show_input(["id"])
                    item_id = int(input())
                if operation == 1:

print(self.model.get_film_and_categories(item_id))
                    input()
                    return
                elif operation == 2:
```

```
print(self.model.get_reviews_to_film(item_id))
                    input()
                    return
                elif operation == 3:
                    print(self.model.get_users())
                    input()
                    return
                elif operation == 4:
                    print(self.model.get_films())
                    input()
                    return
                elif operation == 5:
                    print(self.model.get_categories())
                    input()
                    return
                elif operation == 0:
                    return
                else:
                    self.view.clear()
                    continue
        except:
            print("Bad input data")
            input()

    def show_insert(self, operation):
        self.view.clear()
        try:
            if operation == 1:
                self.view.show_input(["username", "password",
"avaUrl"])
                print(self.model.add_user(input(), input(),
input()))
                input()
            elif operation == 2:
                self.view.show_input(["name", "director"])
                print(self.model.add_film(input(), input()))
                input()
            elif operation == 3:
                self.view.show_input(["name"])
                print(self.model.add_category(input()))
                input()
            elif operation == 4:
                self.view.show_input(["comment", "rating",
"user_id", "film_id"])
                print(self.model.add_review(input(),
int(input()), int(input()), int(input())))
                input()
            elif operation == 5:
                self.view.show_input(["film_id", "category_id"])
                print(self.model.add_film_category(int(input()),
int(input())))
                input()
            elif operation == 0:
                return
            else:
                return
        except:
            print("Bad data")
            input()
            return

    def show_update(self, operation):
        self.view.clear()
        try:
            if operation == 1:
                self.view.show_input(["id", "username",
"password", "avaUrl"])
                print(self.model.edit_user(int(input()), input(),
input(), input()))
                input()
            elif operation == 2:
                self.view.show_input(["id", "name", "director"])
                print(self.model.edit_film(int(input()), input(),
input()))
                input()
            elif operation == 3:
                self.view.show_input(["id", "name"])
                print(self.model.edit_category(int(input()),
input()))
                input()
            elif operation == 4:
                self.view.show_input(["id", "comment", "rating",
"user_id"])
                print(self.model.edit_review(int(input()),
input(), int(input()), int(input())))
                input()
            elif operation == 0:
                return
            else:
                return
        except:
            print("Bad data")
            input()
            return

    def show_remove(self, operation):
        self.view.clear()
        try:
            if operation == 1:
                self.view.show_input(["id"])
                print(self.model.remove_user(int(input())))
                input()
            elif operation == 2:
                self.view.show_input(["id"])
                print(self.model.remove_film(int(input())))
                input()
            elif operation == 3:
                self.view.show_input(["id"])
                print(self.model.remove_category(int(input())))
                input()
            elif operation == 4:
                self.view.show_input(["id"])
                print(self.model.remove_review(int(input())))
                input()
            elif operation == 0:
                return
            else:
                return
        except:
            print("Bad data")
            input()
            return

    def show_special_search(self):
        self.view.clear()
        try:
            self.view.show_special_search()
            print(self.model.special_find(input(), input(),
input()))
            input()
            return
        except:
            print("Something wrong")
            input()
            return
```