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»**

**Тема**: «Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL»

Виконав:   
студент групи КВ-81

Путієнко Сергій  
Перевірив:  
Петрашенко А.В.

Київ 2020

*Метою роботи* є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL  
  
  
 **Завдання**

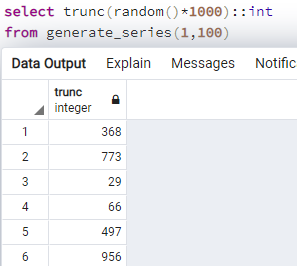
*Загальне завдання* роботи полягає у наступному:

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 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).

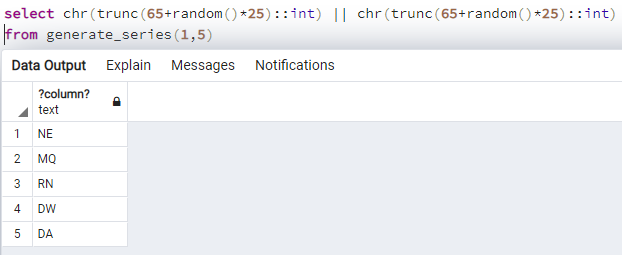
*Деталізоване завдання*:

1. Забезпечити можливість уведення/редагування/вилучення даних у таблицях бази даних з можливістю контролю відповідності типів даних атрибутів таблиць (рядків, чисел, дати/часу). Для контролю пропонується два варіанти: контроль при введенні (валідація даних) та перехоплення помилок (try..except) від сервера PostgreSQL при виконанні відповідної команди SQL. Особливу увагу варто звернути на дані таблиць, що мають зв’язок 1:N. При цьому з боку батьківської таблиці необхідно контролювати **вилучення** рядків за умови наявності даних у підлеглій таблиці. З точки зору підлеглої таблиці варто контролювати наявність відповідного рядка у батьківській таблиці при виконанні **внесення** нових даних. Унеможливити виведення програмою системних помилок на екрані шляхом їх перехоплення і адекватної обробки. Внесення даних виконується користувачем у консольному вікні програми.
2. Забезпечити можливість автоматичної генерації великої кількості даних у таблицях за допомогою вбудованих у PostgreSQL функцій роботи з псевдовипадковими числами. Дані мають бути згенерованими **не мовою програмування, а відповідним SQL-запитом**!

Приклад генерації 100 псевдовипадкових чисел:



Приклад генерації 5 псевдовипадкових рядків:



Приклад генерації псевдовипадкової мітки часу з діапазону [доступний за посиланням](https://stackoverflow.com/questions/22964272/postgresql-get-a-random-datetime-timestamp-between-two-datetime-timestamp/22965061).

Кількість даних для генерування має вводити користувач з клавіатури. Для тесту взяти 100 000 записів для однієї-двох таблиць.

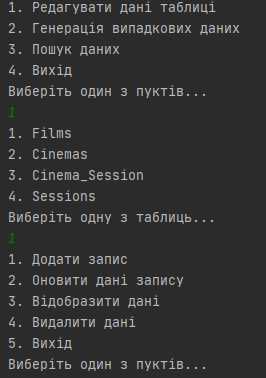
Особливу увагу слід звернути на відповідність даних вимогам зовнішніх ключів з метою уникнення помилок порушення обмежень цілісності (foreign key).

1. Для реалізації пошуку необхідно підготувати 3 запити, що включають дані з декількох таблиць і фільтрують рядки за 3-4 атрибутами цих таблиць. Забезпечити можливість уведення конкретних значень констант для фільтрації з клавіатури користувачем. Крім того, після виведення даних необхідно вивести час виконання запиту у мілісекундах. Перевірити швидкодію роботи запитів на попередньо згенерованих даних.
2. Програмний код організувати згідно шаблону Model-View-Controller(MVC).  Приклад організації коду згідно шаблону доступний [за даним посиланням](https://www.giacomodebidda.com/mvc-pattern-in-python-introduction-and-basicmodel/). При цьому модель, подання та контролер мають бути реалізовані у окремих файлах. Для доступу до бази даних використовувати **лише мову SQL** (без ORM).

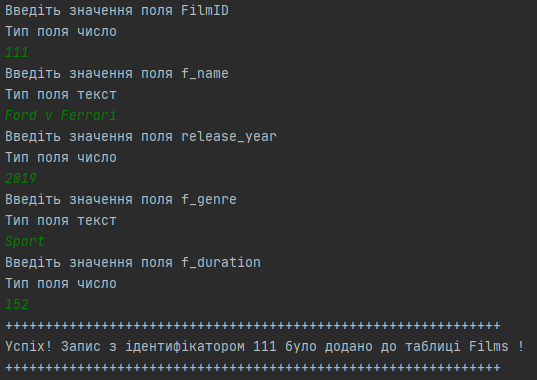
Рекомендована бібліотека взаємодії з PostgreSQL Psycopg2: http://initd.org/psycopg/docs/usage.html)

**Завдання 1**

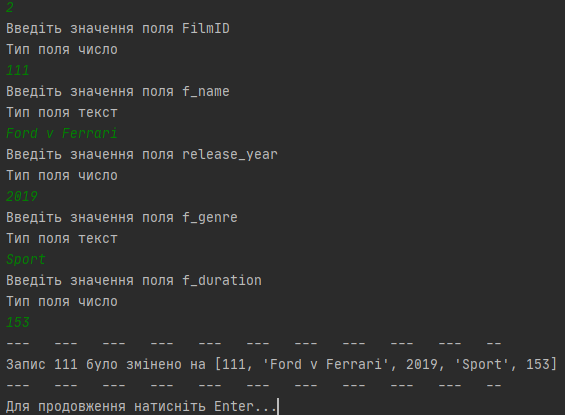
Дана програма дозволяє можливість уведення, редагування, вилучення або перегляду даних вибраної таблиці.



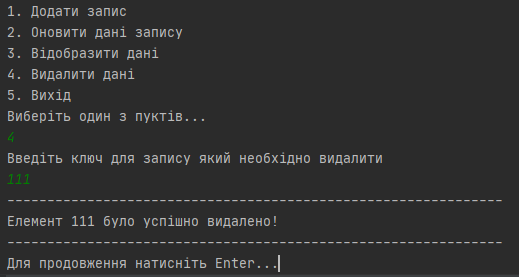
Програма читає дані типів з бази даних і пропонує користувачу ввести відповідні дані. Приклад введення нових даних запису до таблиці Films.



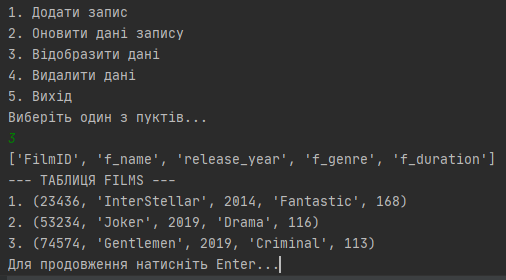
Оновлення даних таблиці Films:



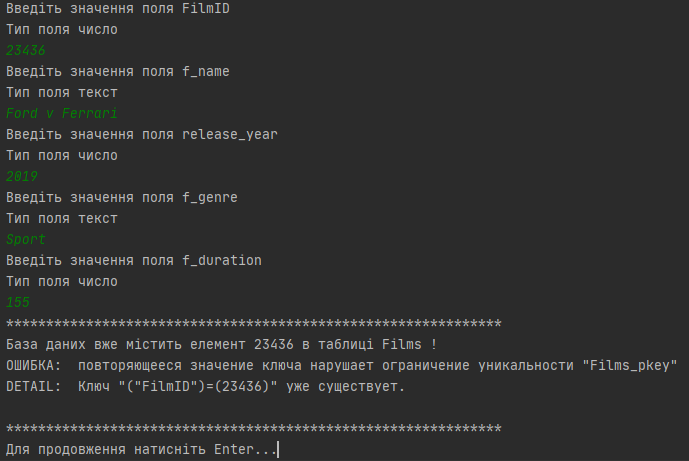
Видалення запису з таблиці Films:



Перегляд записів:

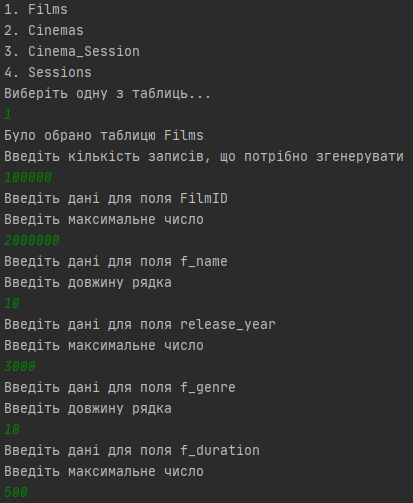


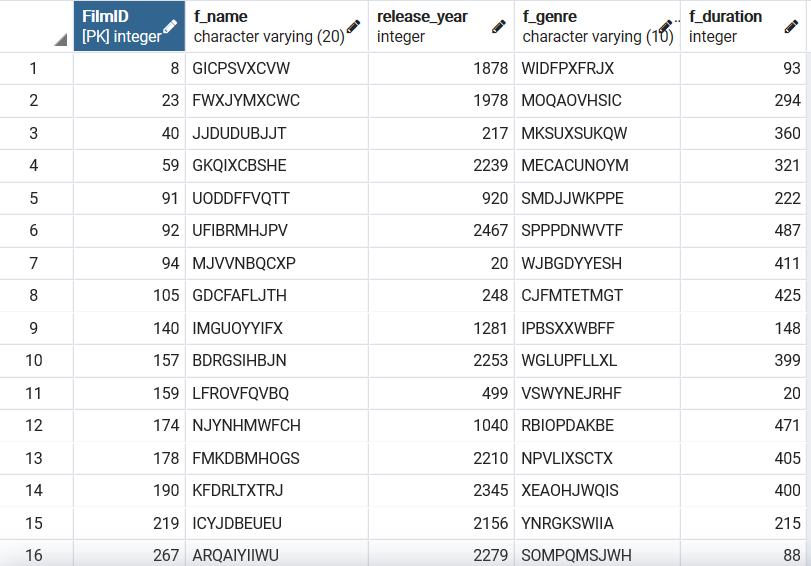
Приклад введення запису з ключем, що вже існує у базі даних:



**Завдання 2**

Дана програма передбачає створення числових даних, рядкових даних або даних, що відповідають датам. Приклад вводу даних для генерації записів у таблиці Films.



Результати:  


Запити:

SELECT trunc(random()\*2000000)::int from generate\_series(1,100000)

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) || chr(trunc(65 + random()\*25)::int) from generate\_series(1,100000)

SELECT trunc(random()\*3000)::int from generate\_series(1,100000)

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) || chr(trunc(65 + random()\*25)::int) from generate\_series(1,100000)

SELECT trunc(random()\*500)::int from generate\_series(1,100000)

**Завдання 3**

У програмі реалізовано три запиту пошуку:

1. Фільми, що будуть показувати після дати у кінотеатрі  
2. Фільми, що будуть показувати після дати та тривалістю у рамках заданого  
3. Фільми, назви яких містять словосполучення та відповідного жанру

Перші два використовують дані з чотирьох таблиць, а третій запит виконує пошук по таблиці фільми. Також запити виконують фільтрацію за чотирьома атрибутами.

**Запит 1.**

SELECT f\_name, f\_genre, start\_date, hall\_name ' \  
 'FROM "Films", "Sessions", "Cinema\_Session", "Cinemas" ' \  
 'WHERE "Films"."FilmID" = "Sessions"."FilmID"' \  
 'AND "Sessions"."SessionID" = "Cinema\_Session"."SessionID"' \  
 'AND "Cinema\_Session"."CinemaID" = "Cinemas"."CinemaID"' \  
 'AND "Cinemas".c\_name = \'{}\' AND "Sessions".start\_date > \'{}\' '

**Запит 2.**

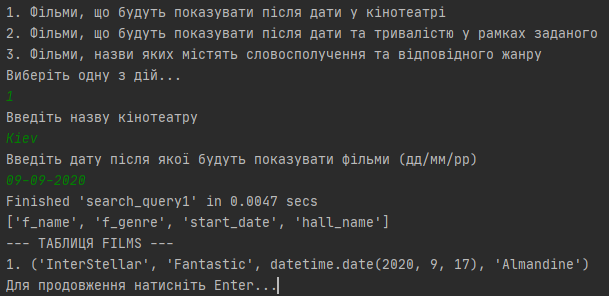
'SELECT f\_name, f\_duration, start\_date, c\_name ' \  
 'FROM "Films", "Sessions", "Cinema\_Session", "Cinemas" ' \  
 'WHERE "Films"."FilmID" = "Sessions"."FilmID" ' \  
 'AND "Sessions"."SessionID" = "Cinema\_Session"."SessionID" ' \  
 'AND "Cinema\_Session"."CinemaID" = "Cinemas"."CinemaID" ' \  
 'AND "Sessions"."start\_date" > \'{}\' ' \  
 'AND "Films".f\_duration BETWEEN {} AND {}'

**Запит 3.**

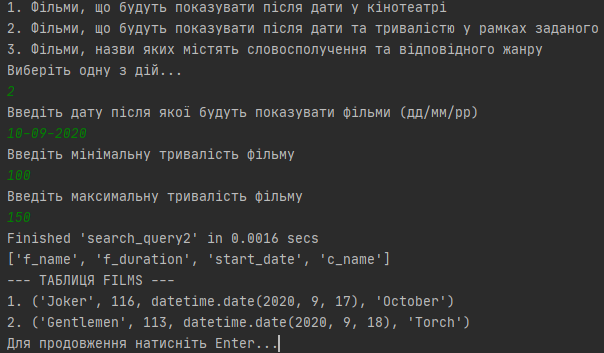
'SELECT f\_name, f\_genre, f\_duration, release\_year ' \  
 'FROM "Films"' \  
 'WHERE ' \  
 '"Films".f\_name LIKE \'%{}%\' ' \  
 'AND "Films".f\_genre LIKE \'%{}%\' '

Далі наведено результати роботи програми з використанням відповідних запитів.

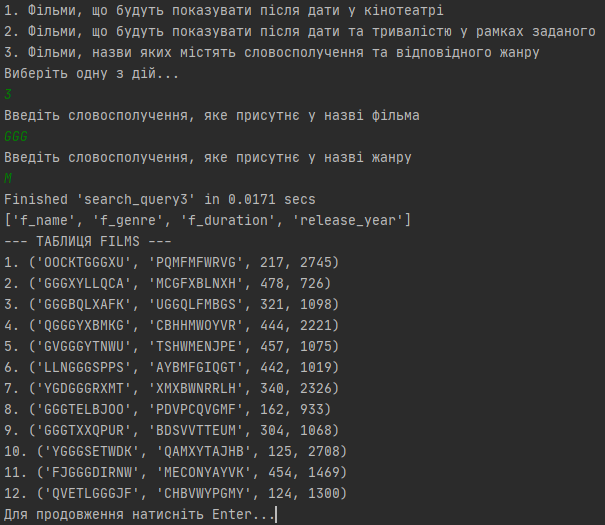
Перший запит:



Другий запит:



Третій запит:



**Завдання 4**

Код програми знаходиться у репозиторію github за посиланням:  
  
<https://github.com/CrispBS/Databases-and-controls.git>  
  
  
