Movie Database

ECE 59500

Chris Collier

Vincent Crabtree

Daniel Kobold

Abstract

The movie database is a complete system that stores movie information including movie name, director, year and genre. It also allows for both regular users and critics to write reviews and add ratings to movies which will be displayed on the movie information page. Users are also able to add other users as friends, and movies will be recommended to them according to how their friends rated them. Admins are special users who curate the system by adding movies and deleting movies, and also deleting reviews that are deemed inappropriate or invalid. The system is accessed through a user interface where users log on and browse movie and user information. It is through this interface that users and critics can make ratings and reviews.

The database system is comprised of three main components: A Python program for the user interface and back-end, a Microsoft SQL Server database for storing user, critic, movie and ratings information, and a MongoDB NoSQL database to store user friends lists and movie reviews. The Python program acts as an interface between the user and the data, and converts user input into queries that each database can understand. The TkInter library is used to create a graphical user interface (GUI) that displays movie information in a neat window for the user. It also includes a search bar and buttons that allows users to enter search queries and select movies from a list. The pyodbc library is used to interface the GUI with the MS SQL database. This library allows the user input to be added to a SQL query and then executes that query on the database. Finally, PyMongo was the chosen library for interfacing between the GUI and the MongoDB database. This library makes it easy to execute NoSQL queries from a Python program.

The MS SQL database stores movie, user and ratings information in various tables that store have specific relationships with each other. There are five tables in total: a table each for users and critics, a table each for user and critic reviews and a table for movies. The MongoDB database is used to store friends lists for each user, as well as the reviews for each movie. The documents for each of these items are stored in a completely independent manner, and they have no direct relationships with each other. The system Admins are responsible for adding movies into the SQL database and for removing ratings or reviews from either database that if they seem inappropriate or misplaced.

Introduction

Finding a movie review in today’s day and age is about as simple as it comes. Easily over 600 movies hit the US box offices each year, and that number continues to rise. In a world seemingly driven by social media and an online presence, having the ability to broadcast to the entire world at the tip of your fingers is no longer a thing of the future. There is no short supply of means to access a movies information or inquire about what others thought about a movie. What is lacking, is a method and system to share thoughts and opinions about movies with friends in a social media setting. The current existing systems all showcase the newest movies with ratings, raves, criticisms and critiques. Additionally, there are systems devoted to the lesser important details such as underlying actors and various locations the movie may have been filmed. There is also no lack of platforms to share one’s opinion on movies and unrelated thoughts alike. This system attempts to bridge the divide between these two well established worlds by allowing the patrons to write their own reviews, give their own ratings, see reviews of established movie critics, receive future recommendations, retrieve specific movie information, and most importantly connect with their friends.

Literary Survey

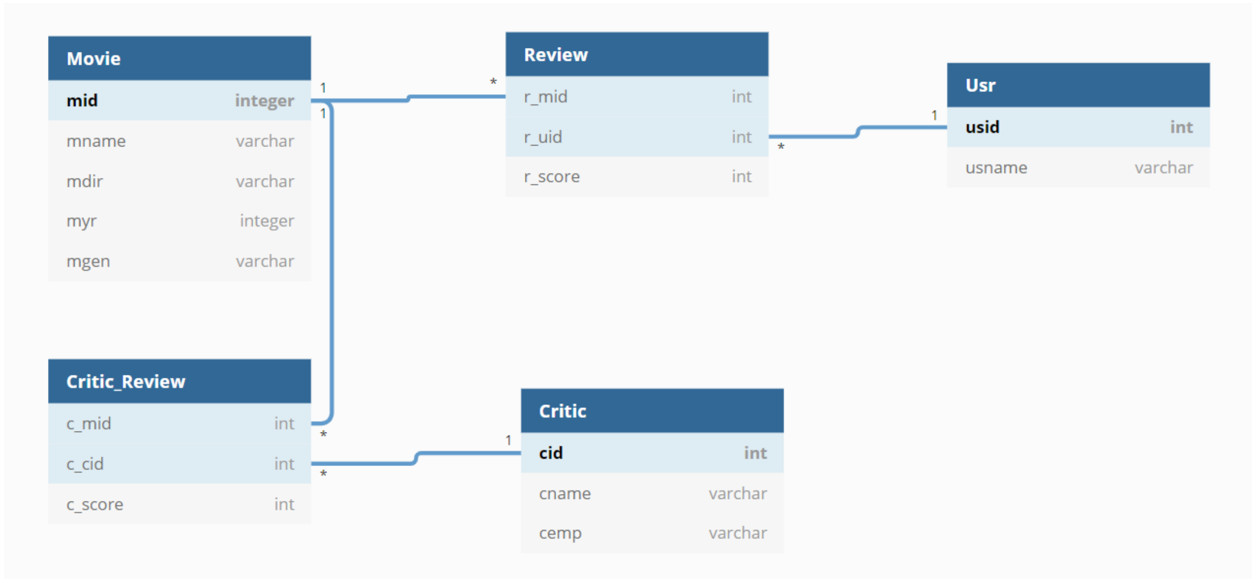
Of the currently established systems one tends to rise above and rightfully holds its name and place. The International Movie Database (IMDb) is an online database that holds information related to films, television shows, and videogames. On the top level, IMDb showcases the major details such as the full cast, a storyline overview and summary, the title genre, Motion Picture Rating, and media trailers and pictures. Items like the technical specifications (e.g. the runtime, sound mix, color, aspect ratio and title format) can also be found. The database also exhibits trivia about the object, popular quotes, and even some frequently asked questions. Outside of the title information IMDb has boasted many new features over its development. Select titles can be streamed and viewed directly from the website. For the movies and films that are actively in theatres, showtimes and viewing locations are displayed. A ticket can also be purchased. This is accomplished through a redirect to “fandango.com”, a popular movie ticketing service. For a more culture route, the website tracks and displays information about celebrities, events, and community news. Celebrity information consists of recent photos, links to news articles and a Born Today sections to catalog those whose birthday it is. News articles cover a wide range of topics such as movies, TV, Indie, and again celebrity updates. By creating an account, one is allowed access to personal viewing recommendations, a Watchlist to track movies to see, and the ability to write and record your own ratings and reviews.

Rotten Tomatoes, one of IMDb’s largest competitors, demonstrates a familiar layout to the items previously listed. The only significant difference is that the whole platform is based on a system named the Tomatometer®. If a title is seen as favorable or enjoyable, it is given a ‘fresh’ rating. With the higher the freshness the better received the title is. Contrarily if a title is ill received it will obtain a rotten tomato score. These values are based on a zero to 100 integer percentage and are actively updated base on critic and user opinions.

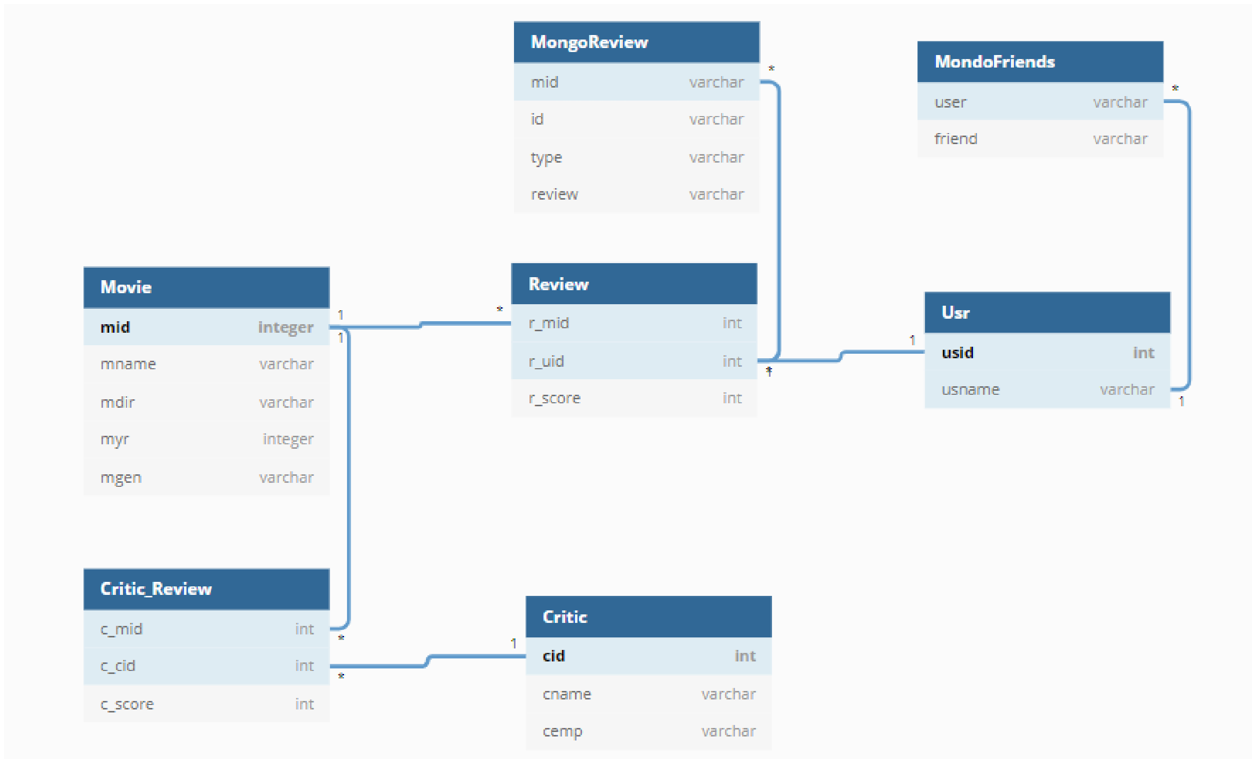
From the early days of online chatrooms and “sixdegrees.com”, to the birth of myspace, Wikipedia, Facebook and Twitter, social media has slowly planted a home within our lives that it does not intent to ever vacate. The idea of sharing information, ideas, interests and other forms of expression through online networks is no longer a foreign concept, but seemingly the foundation for our everyday lives. Roughly one third of all people on earth have used Facebook, a squandering static when only 130 million users take to Twitter daily. Regardless, In the United States and throughout most of the world, these two platforms dominate the online social media market. Both incorporate the ability to share pictures, videos, external links, thoughts and opinions, all at the cost of a few button presses. These platforms handle seemingly everything well, and that’s the problem here. You can find specific subsections devoted strictly to movies and films, but that is not Facebook’s nor Twitter’s main focus. You would be able to view a friend or stranger’s review of a movie, but you would have to specifically be looking for it, and in doing so you would likely have to search through a multitude of over posts before it was found. The goal of providing one, central and organized location for both reviews of a movie, and a social aspect around movies is the scope of this project.

System Design

ER Diagram



ER Diagram with MongoDB Document Structure Design



Methodology

MS SQL Server

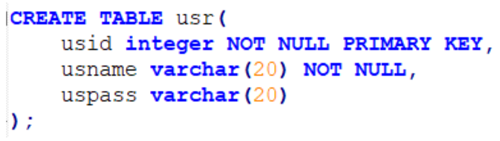
Microsoft SQL Server is used to store a large percentage of the movie and user data for the system. The SQL Server database includes a table for both users and critics. It also includes a table for movies each type of movie review. As shown in Figure 2, the movies table has a one-to-many relationship with both the critic and user review tables. This means that each movie can have multiple reviews written about it but each review can only be about one movie. These review tables have a many-to-one relationship with the different types of users, meaning that each review can only be written by one user but each user can write many reviews. Although the MongoDB collections aren't directly linked to the SQL tables, each review in MongoDB corresponds to a review in SQL and each friends list is liked to a user.

Admin

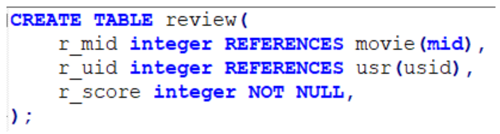
Admins are implemented in the python program for the project. They have a unique log-in and are responsible for adding or removing movies from the system. They are the only type of user in the system that can use the "create table" and "delete" queries.

User

Users are the default type of user in the system. They are the normal users who can search for movies and read/write ratings and reviews. Users also have a friends list stored in MongoDB that is linked to them using their user id. Users can use the user interface to call a select query to search for movies or other users in the system. They can also use an instert statement to create ratings for movies.



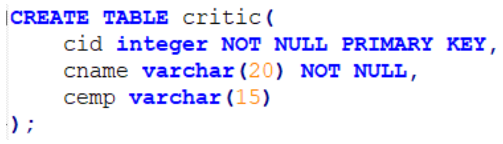
```
CREATE TABLE usr(
    usid integer NOT NULL PRIMARY KEY,
    usname varchar(20) NOT NULL,
    uspass varchar(20)
);
```



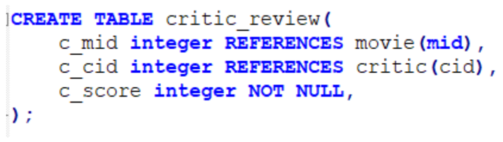
```
CREATE TABLE review(
    r_mid integer REFERENCES movie(mid),
    r_uid integer REFERENCES usr(usid),
    r_score integer NOT NULL,
);
```

Critic

Critics are similar to users, but write special reviews that are shown separately from normal user reviews called critic reviews. They have the same abilities as normal users, but cannot have a friends list because their only function is to write critic reviews.



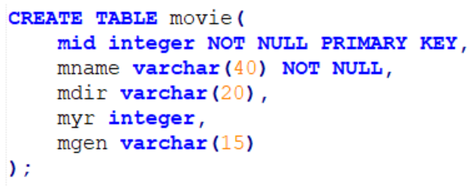
```
CREATE TABLE critic(
    cid integer NOT NULL PRIMARY KEY,
    cname varchar(20) NOT NULL,
    cemp varchar(15)
);
```



```
CREATE TABLE critic_review(
    c_mid integer REFERENCES movie(mid),
    c_cid integer REFERENCES critic(cid),
    c_score integer NOT NULL,
);
```

Movie storage system

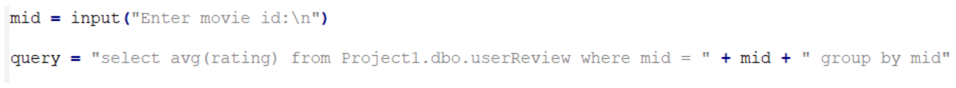
The movie table is used to store the various movies that are entered into the database. The only type of user that can add or remove entries from this table is the Admin.



```
CREATE TABLE movie(
    mid integer NOT NULL PRIMARY KEY,
    mname varchar(40) NOT NULL,
    mdir varchar(20),
    myr integer,
    mgen varchar(15)
);
```

Average Rating

In order to show the average ratings for each movie in the user interface, the SQL select query below was utilized. The query takes the movie id as input and returns the average of all ratings for that id.



```
mid = input("Enter movie id:\n")

query = "select avg(rating) from Project1.dbo.userReview where mid = " + mid + " group by mid"
```

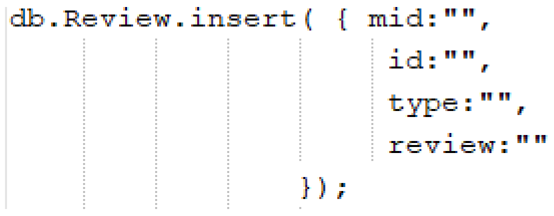
MongoDB

MongoDB was used to house and handle the non-relational aspects of the project. These items include the reviews of both critics and users, and the friends of each user.

Inside the database there are two collections, Review and Friends.

Review

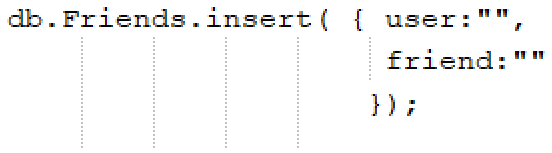
The Review collection stores the physical text of a review from either a review or a critic. Both reviews from a user and a critic are stored in the same collection and each review is stored as a new document. They are differentiated by the field ‘type’, where this will be ‘User’ for a user and ‘Critic’ for a critic. For either case, the corresponding movie ID, the ID of the specific reviewer, and the review text are stored in the fields mid, id, and review respectfully.



```
db.Review.insert( { mid:"",
                     id:"",
                     type:"",
                     review:""
                 });
```

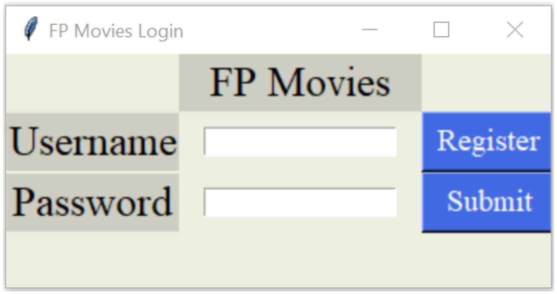
Friends

The Friends Collection stores the friends of a specific user. Each friend is stored as a new document. The field ‘user’ is the specific user who is logged in to the system. Upon each addition of a friend, the friend who is being added, their username is stored in the field ‘friend’.

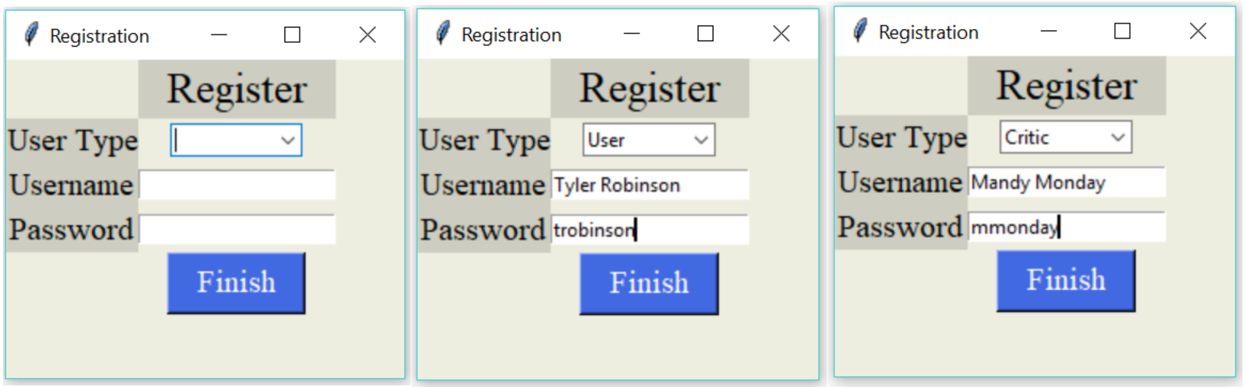


```
db.Friends.insert( { user:"",
                      friend:""
                    });
```

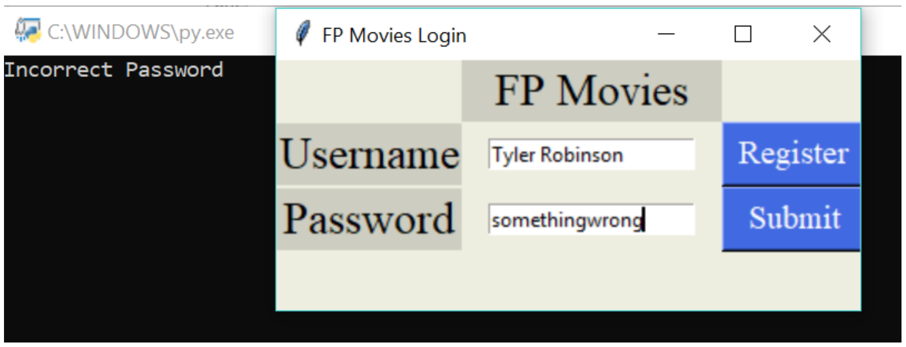
Graphical User Interface



Upon clicking the python file or executing from the command line, the above window appears. The option buttons will either launch the registration window or will submit the login information on the left side of the window, depending on whether "Register" or "Submit" is clicked.

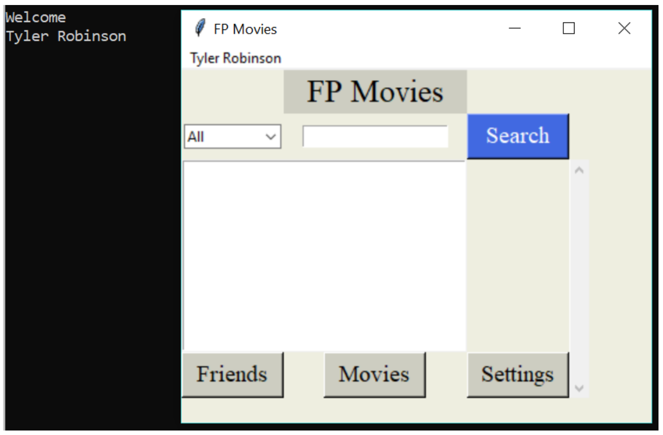


The "Register" button will launch the window above and to the left. There is a dropdown box that has the choices "User" and "Critic", which can be selected depending on whether the person registering is a general user or a movie critic. The two windows above and to the right show sample input for each type of user. These accounts were both finalized by clicking the "Finish" buttons, and will be used to demonstrate other features of the program.



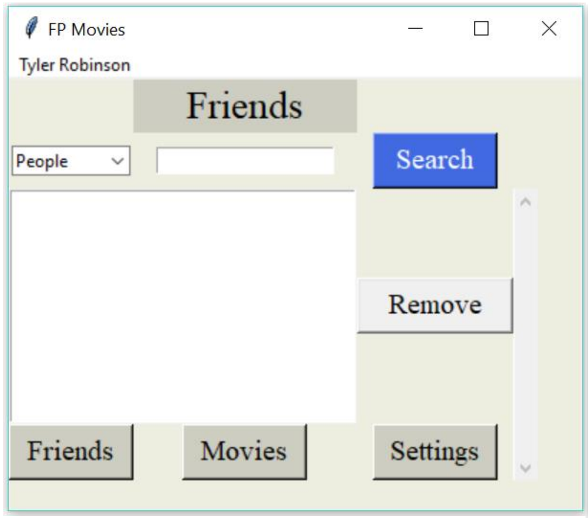
```
Incorrect Password
```

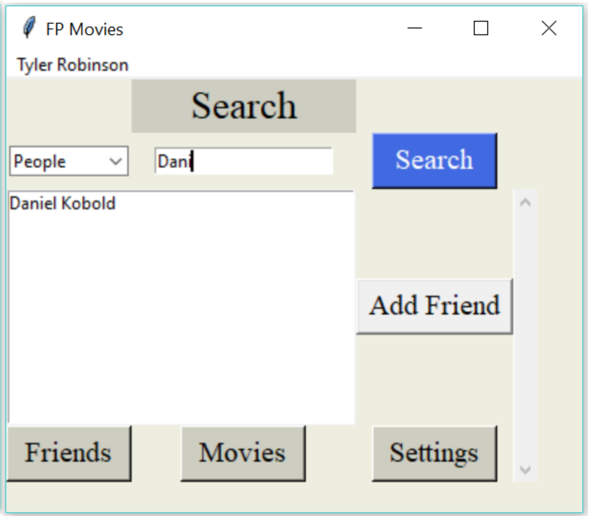
When attempting to login, typing an incorrect username or password and clicking “Submit” will result in an “Incorrect Password” message appearing in the console window, and will not allow the user to proceed into the program.



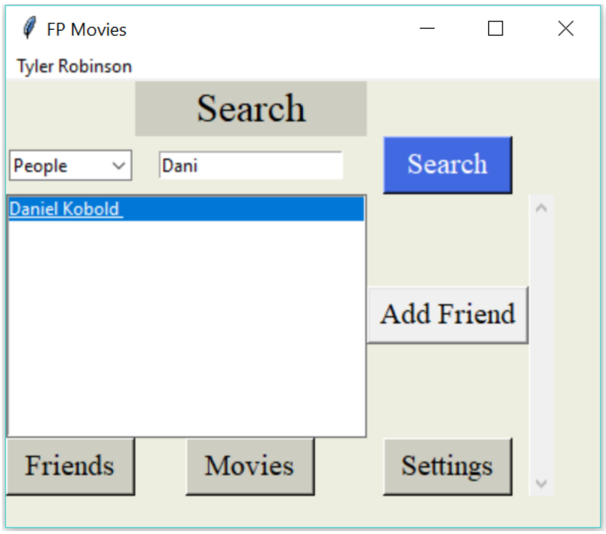
```
Welcome
Tyler Robinson
```

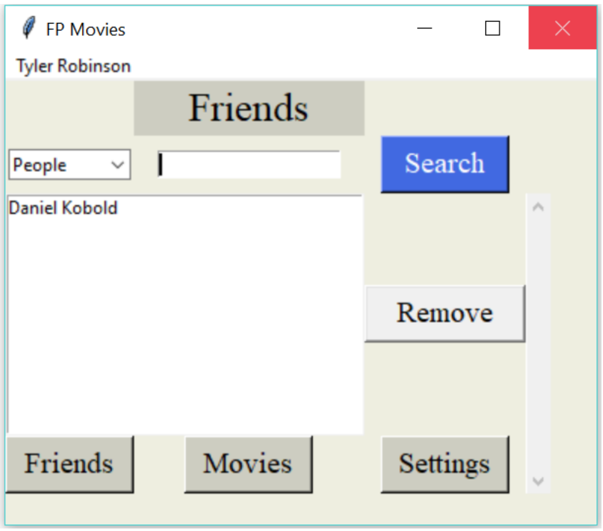
Typing a correct username and password pair will close the login window, open the FP Movies window, and will display a welcome message in the console window. The FP Movies window has many options. The user’s name is shown in the top left corner, and the title of the window is shown as well. The dropdown on the left side gives some options for the search bar where the search criteria can be narrowed to People, Movies, or Settings, much like the buttons at the bottom of the screen.



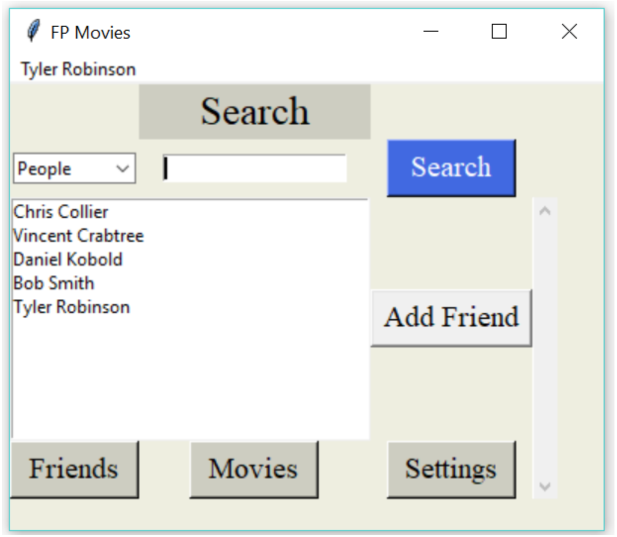


The “Friends” button will list the users connected as “Friends” and will allow the removal of friends. Currently, Tyler Robinson does not have any friends as this is a brand-new account. To add friends, Tyler will type something such as “Dani” and click Search, upon which my name will appear because “Dani” is in the name “Daniel”.



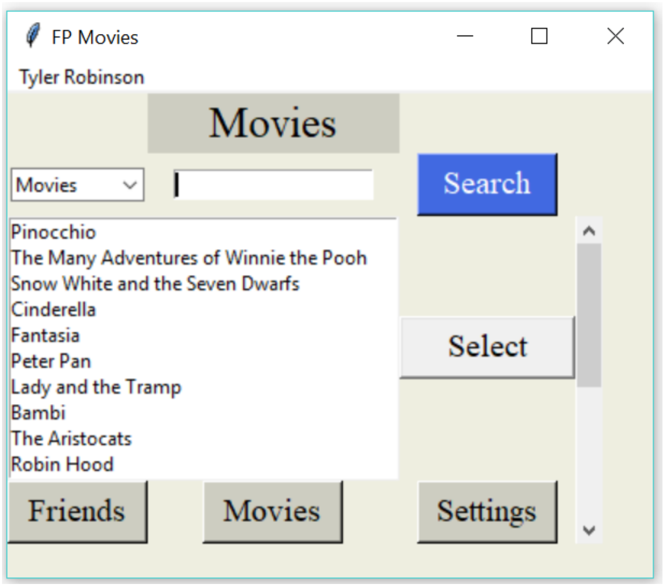


By clicking “Daniel Kobold” in the window, the name will become highlighted in blue and I can be added as Tyler’s friend. After clicking “Add Friend”, the window will switch to the Friends List and will show my name as a part of the list. This process adds an entry to the Friends table in MongoDB with the user specified as Tyler and the friend specified as Daniel. Because of the way it is formatted, when I (Daniel) click on my friends list, Tyler will not appear unless I add him. In this way, the system is more of a “follow” system than a “friends” system, but the name was kept as friends to make the system feel more personal to each user.



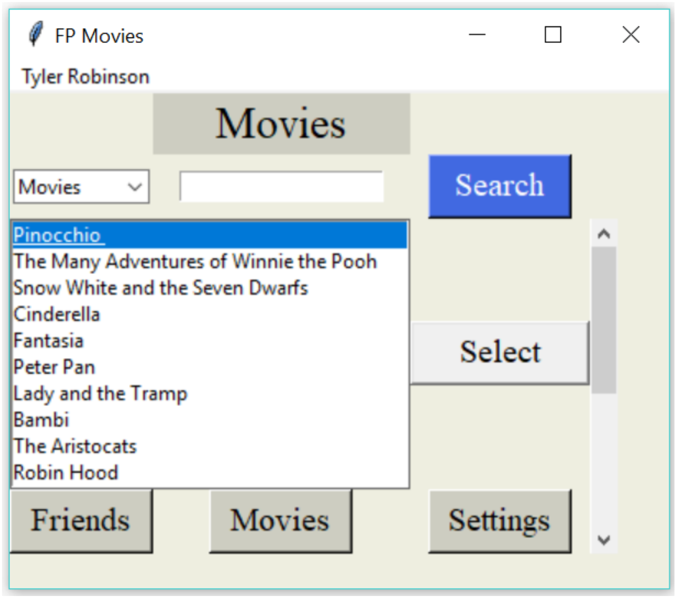
Chris Collier
Vincent Crabtree
Daniel Kobold
Bob Smith
Tyler Robinson

It is also worth noting that emptying the search bar and clicking “Search” will list all the users currently in the system. However, the list will not include the Critics because they are considered another user type. Notice that our critic example “Mandy Monday” does not appear in the list of people.



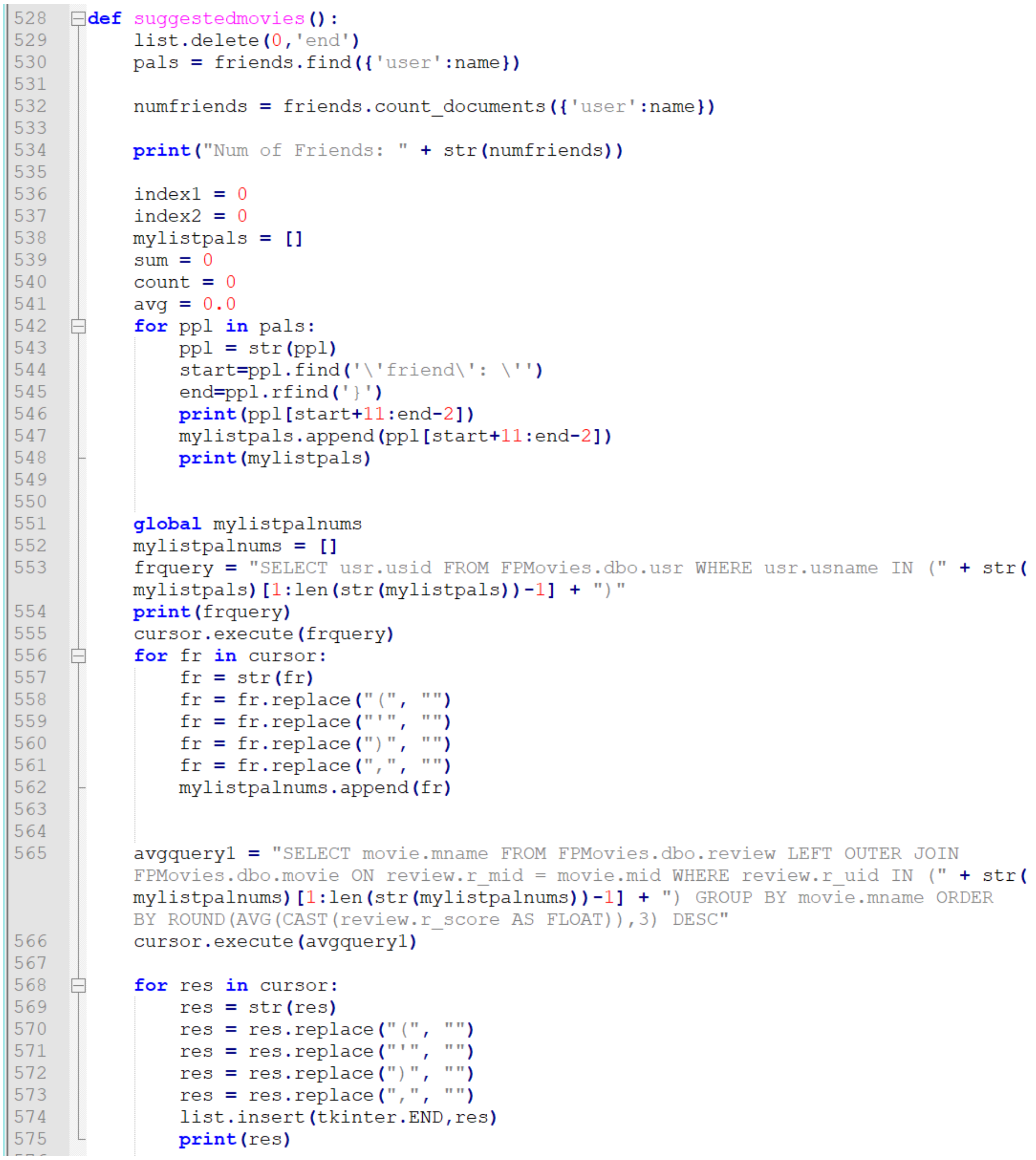
Pinocchio
The Many Adventures of Winnie the Pooh
Snow White and the Seven Dwarfs
Cinderella
Fantasia
Peter Pan
Lady and the Tramp
Bambi
The Aristocats
Robin Hood

With a friend added to his friends list, Tyler now clicks on the "Movies" button, which will list some movies. The movies that are listed and the order they are listed in is set by a multi-query section that determines movies that have been reviewed by Tyler's friends and ranks them by the average of the scores that his friends have given the movie. This part involved a NOSQL query to determine the users on the user's friend list, followed by a long query to average the scores of the relevant users. The code can be found on the next page, along with an explanation of how it works.



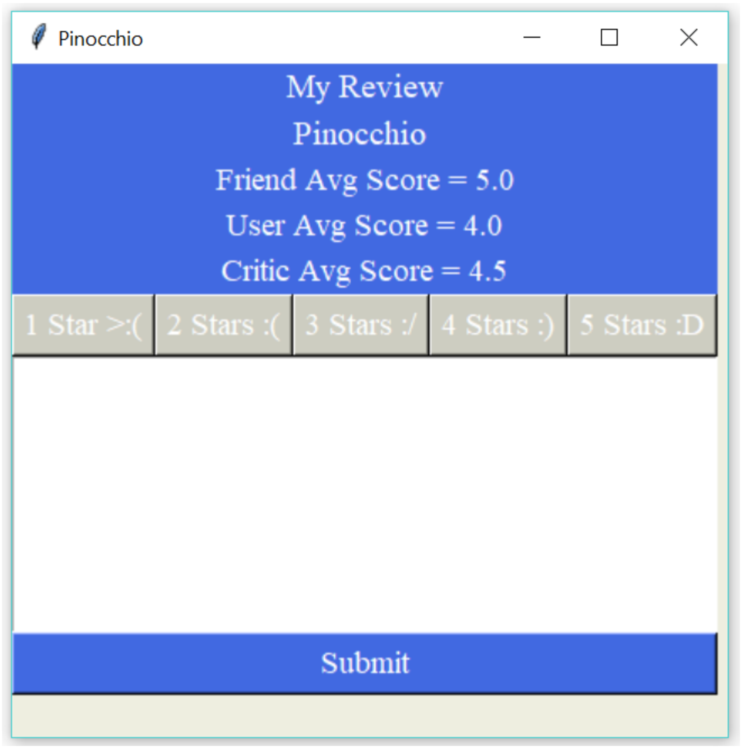
Pinocchio
The Many Adventures of Winnie the Pooh
Snow White and the Seven Dwarfs
Cinderella
Fantasia
Peter Pan
Lady and the Tramp
Bambi
The Aristocats
Robin Hood

Once again, Tyler can click on a movie and it will be highlighted in blue. Clicking select will keep this window open, but will also launch the review window, which will be covered after the code for the suggested movies algorithm.



```python
def suggestedmovies():
    list.delete(0,'end')
    pals = friends.find({'user':name})

    numfriends = friends.count_documents({'user':name})

    print("Num of Friends: " + str(numfriends))

    index1 = 0
    index2 = 0
    mylistpals = []
    sum = 0
    count = 0
    avg = 0.0
    for ppl in pals:
        ppl = str(ppl)
        start=ppl.find('\'friend\': \'')
        end=ppl.rfind('}')
        print(ppl[start+11:end-2])
        mylistpals.append(ppl[start+11:end-2])
        print(mylistpals)


    global mylistpalnums
    mylistpalnums = []
    frquery = "SELECT usr.usid FROM FPMovies.dbo.usr WHERE usr.usname IN (" + str(mylistpals)[1:len(str(mylistpals))-1] + ")"
    print(frquery)
    cursor.execute(frquery)
    for fr in cursor:
        fr = str(fr)
        fr = fr.replace("(", "")
        fr = fr.replace("'", "")
        fr = fr.replace(")", "")
        fr = fr.replace(",", "")
        mylistpalnums.append(fr)


    avgquery1 = "SELECT movie.mname FROM FPMovies.dbo.review LEFT OUTER JOIN FPMovies.dbo.movie ON review.r_mid = movie.mid WHERE review.r_uid IN (" + str(mylistpalnums)[1:len(str(mylistpalnums))-1] + ") GROUP BY movie.mname ORDER BY ROUND(AVG(CAST(review.r_score AS FLOAT)),3) DESC"
    cursor.execute(avgquery1)

    for res in cursor:
        res = str(res)
        res = res.replace("(", "")
        res = res.replace("'", "")
        res = res.replace(")", "")
        res = res.replace(",", "")
        list.insert(tkinter.END,res)
        print(res)
```

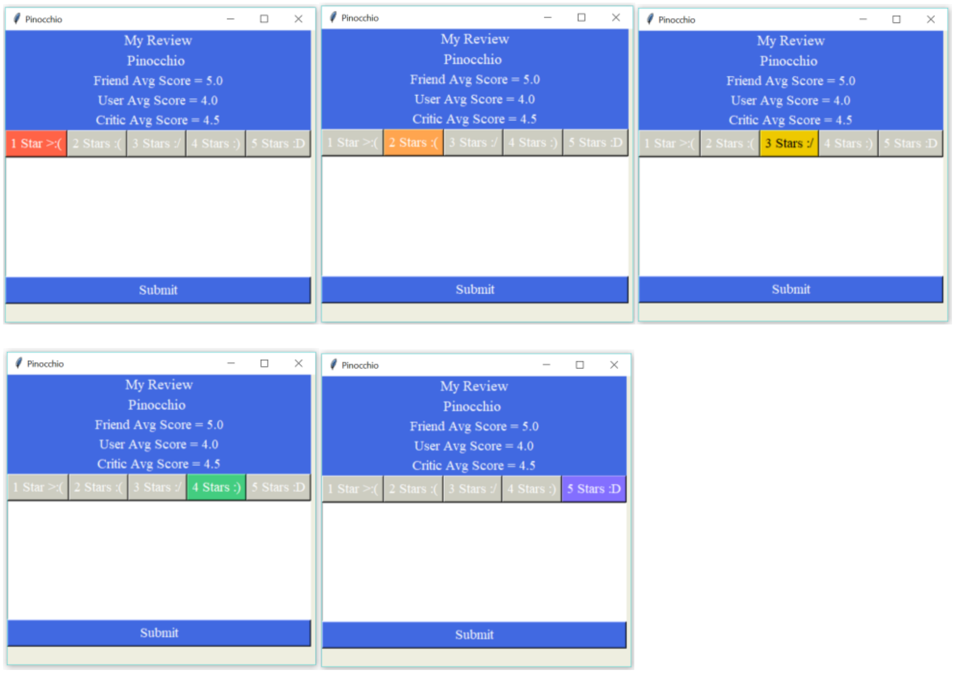
The code for the above operation is shown above. First, the window is cleared and then the user's friends are found by looking in the Friends table of the NOSQL database. The number of friends is determined, and a list of friend names is made. Next, that list is used to find the user id numbers for the friends on the list, based on the user table in the SQL database. These numbers are then used when selecting the relevant reviews from the movie table, which are then used to find the corresponding score and to sort the movies by the average score. Finally, the movies are displayed on the screen in order by the average of the scores given by the user's friends.



As mentioned previously, by clicking the “Select” button, the review window will appear. Underneath the movie title, three scores will appear for the user. The first will show the average score given to this movie by Tyler’s friends, which is also the criteria of the order by which the movies were listed in the previous window. The second will show the average score of all the users of the database. And finally, the third score will show the average score given to the movie by the critics. The above is a good example because it shows three different scores for the different categories of users.

The window above shows a few things. First, about halfway down, the star rating buttons are shown. These specify the score from 1-5 that the user wants to rate the movie, along with a small ASCII face representing a possible emotion that the viewer felt when watching the movie. Clicking one of these will highlight the selected button in a certain color depending on how high of a rating is given (see the next page). Each time one of these is clicked, the SQL database is updated with the new star rating, or if there is currently not a rating in the database, a new entry is added.

The white area on the box is the text entry area where the user can write comments and thoughts about the movie. This part will not be updated until the user clicks “Submit”, at which time the window will disappear. The text section is stored in MongoDB (in the NOSQL database). Even after the review window is closed by clicking “Submit”, the same information will reappear when that movie is selected again.


Pinocchio

My Review

Pinocchio

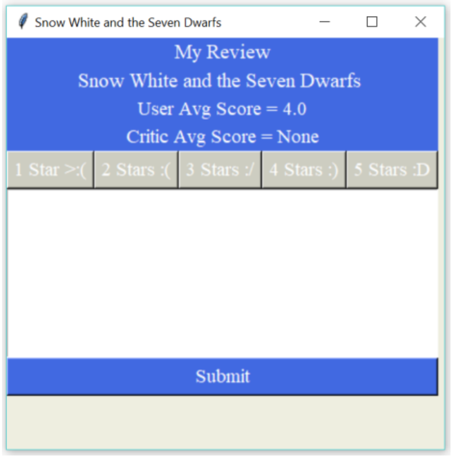
Friend Avg Score = 5.0

User Avg Score = 4.0

Critic Avg Score = 4.5

The above images show the colors that the buttons will turn after being clicked. By clicking another button, the rest of the buttons turn back to gray to indicate that they are not the selected rating. Once again, this choice, along with any comments made in the text box below, will appear again when the movie is selected after this session ends and the user clicks "Submit".

This concludes the main features of the User. The Critic actions are all very similar to the User actions, but all reviews are separated from the user reviews. Also, critics do not have a friend list, and can only review movies. Having a friends list as a critic could skew their ratings and would make the rating system of the program unfair to users who take critic opinions very seriously.


Snow White and the Seven Dwarfs

My Review

Snow White and the Seven Dwarfs

User Avg Score = 4.0

Critic Avg Score = None

One last difference for critics is the movie review display. Only the user average scores and critic average scores are shown, due to the fact that a critic is not allowed to have friends as explained above.

Conclusions

The movie database system allows users and critics to create and read reviews for movies of all types. The system also includes a social media aspect that allows users to add friends and recommends movies to users based on their friends list. Admins run the system by adding and removing movies from the database. The system consists of three main components: a SQL Server database to house most of the movie and user data, a MongoDB database to store written movie reviews and user friends lists, and a Python program that includes a graphical user interface and turns user input into queries that return information for the user to see.

References

Facebook. (2019). Retrieved from https://www.facebook.com/

Fandango. (2019). Movies | TV Shows | Movie Trailers | Reviews. Retrieved from https://www.rottentomatoes.com/

Graphical User Interfaces with Tk. (n.d.). Retrieved from https://docs.python.org/3/library/tk.html

How to Connect Python to SQL Server using pyodbc. (n.d.). Retrieved from https://datatofish.com/how-to-connect-python-to-sql-server-using-pyodbc/

Imdb. (2019). Ratings and Reviews for New Movies and TV Shows. Retrieved from https://www.imdb.com/

Microsoft. (2019). Unleash the power in your data. Retrieved from https://www.microsoft.com/en us/sql-server/default.aspx

MongoDB. (2019). PyMongo 3.8.0 Documentation. Retrieved from https://api.mongodb.com/python/current/

Tutorial. (n.d.). Retrieved from http://api.mongodb.com/python/current/tutorial.html

Twitter. (2019). Login on Twitter. Retrieved from https://twitter.com/login/

Appendix: Code

fpmovies.py

```
import pyodbc
import tkinter
import tkinter.ttk
from tkinter import scrolledtext
from tkinter import Menu
from tkinter import Listbox
from tkinter import Text
from PIL import ImageTk,Image
import sys
import requests
import pymongo
import io
import base64
from pymongo import MongoClient
from urllib.request import urlopen
import urllib.request
from io import BytesIO

global name                                 #Name of the user
global auth
global list
global selectedmovie
global input
global load
global button
global usrid
global movnum
global newreviewnum                         #1 or 0 depending on if this is a
new review or not
global mylistpalnums
global critic

'''Star Rating Buttons'''
global movie_btn1
global movie_btn2
global movie_btn3
global movie_btn4
global movie_btn5

load = 0
name = ""
auth = 0
critic = 0

'''Set Up Windows'''
#Login window
log = tkinter.Tk()

#Label the window
log.title("FP Movies Login")

#Window size
```

```
log.geometry('350x150')

#Window background color
log.configure(bg='ivory2')

'''Database Connection'''
#Connect to relational database
conn = pyodbc.connect('Driver={SQL Server};'
                                    'Server=DESKTOP-2JG1NIF\SQLEXPRESS;'
                                    'Database=FPMovies;'
                                    'Trusted_Connection=yes;')

#Set up cursor (for displaying text)
cursor = conn.cursor()


'''Non-Relational Database Connection'''
client = MongoClient('localhost',27017)
db = client.FPMovies
friends = db.Friends
rvwtxt = db.Review

#Header label
lbl1 =tkinter.Label(log,text="FP Movies",font=("Times New
Roman",20),bg="ivory3",width = 10)
lbl1.grid(column=1,row=0)

#Log In Screen labels and entry boxes
lblu = tkinter.Label(log,text="Username",font=("Times New
Roman",20),bg="ivory3",width = 7)
lblp = tkinter.Label(log,text="Password",font=("Times New
Roman",20),bg="ivory3",width = 7)
enteruser = tkinter.Entry(log,width=20)
enterpass = tkinter.Entry(log,width=20)
lblu.grid(column=0,row=1)
lblp.grid(column=0,row=2)
enteruser.grid(column=1,row=1)
enterpass.grid(column=1,row=2)

#Complete registration - activated after clicking "Finish" in Register window
def completeregister():
      global reg
      global newuser
      global newpass
      global regcombo
      type = regcombo.get()

      #Insert user/critic info into SQL user/critic table
      if type=="User":
            userquery = "INSERT INTO FPMovies.dbo.usr (usid,usname,uspass)
VALUES ((SELECT TOP 1 usid FROM FPMovies.dbo.usr ORDER BY usid DESC)+1,\'" +
newuser.get() + "\',\'" + newpass.get() + "\')"
            cursor.execute(userquery)
            conn.commit()
      elif type=="Critic":
            criticquery = "INSERT INTO FPMovies.dbo.critic (cid,cname,cpass) 
VALUES ((SELECT TOP 1 cid FROM FPMovies.dbo.critic ORDER BY cid DESC)+1,\'" + 
newuser.get() + "\',\'" + newpass.get() + "\')"
            cursor.execute(criticquery)
            conn.commit()

      #Close this window
      reg.destroy()

#Registration - activated after clicking "Register" in login window
def register():
      global reg
      global newuser
      global newpass
      global regcombo

      #Format tkinter window
      reg = tkinter.Tk()
      reg.title("Registration")
      reg.geometry('250x200')

      #Labels
      lblr = tkinter.Label(reg,text="Register",font=("Times New Roman", 
20),bg="ivory3",width=7)
      lblr.grid(column=1,row=0,columnspan=2,sticky=tkinter.W+tkinter.E+tkinte
r.N+tkinter.S)
      reg.configure(bg="ivory2")
      lblut=tkinter.Label(reg,text="User Type",font=("Times New Roman", 
15),bg="ivory3",width=7)
      lblut.grid(column=0,row=1)

      #Combobox
      regcombo = tkinter.ttk.Combobox(reg,width=10)
      regcombo['values'] = ("User","Critic")
      regcombo.grid(column=1,row=1)

      #Labels and text entry boxes
      lblnu=tkinter.Label(reg,text="Username",font=("Times New Roman", 
15),bg="ivory3",width=7)
      lblnu.grid(column=0,row=2)
      newuser=tkinter.Entry(reg,width=20)
      newuser.grid(column=1,row=2)
      lblnp=tkinter.Label(reg,text="Password",font=("Times New Roman", 
15),bg="ivory3",width=7)
      lblnp.grid(column=0,row=3)
      newpass=tkinter.Entry(reg,width=20)
      newpass.grid(column=1,row=3)

      #Finish registration button
      finishreg=tkinter.Button(reg,text="Finish",bg="royal 
blue",fg="white",font=("Times New Roman", 
15),width=7,command=completeregister)
      finishreg.grid(column=1,row=4)
      reg.mainloop()

#Submit - activated when user clicks submit on login screen
def clickedsubmit():
      global passw
      global name
      global auth

      #Retrieve username and password
      name = enteruser.get()
      passw = enterpass.get()

      #Look up passwords from critic and user tables
      passquery = "SELECT cpass FROM FPMovies.dbo.critic where cname = \'" +
name + "\'"
      cursor.execute(passquery)
      crow = str(cursor.fetchone())
      crow = crow.replace("(", "")
      crow = crow.replace("'", "")
      crow = crow.replace(")", "")
      crow = crow.replace(",", "")
      crow = crow.replace(" ", "")
      passquery = "SELECT uspass FROM FPMovies.dbo.usr where usname = \'" +
name + "\'"
      cursor.execute(passquery)
      row = str(cursor.fetchone())
      row = row.replace("(", "")
      row = row.replace("'", "")
      row = row.replace(")", "")
      row = row.replace(",", "")
      row = row.replace(" ", "")
      global critic

      #Compare to the passwords that match the username entered (if any)
      #Then set critic variable and close window
      if row == passw:
            auth = 1
            print("Welcome")
            critic = 0
            log.destroy()
      elif crow == passw:
            auth = 1
            print("Welcome")
            critic = 1
            log.destroy()
      else:
            print("Incorrect Password")

#Submit and register buttons
btnsubmit = tkinter.Button(log,text="Submit",bg="royal
blue",fg="white",font=("Times New Roman", 15),width=7,command=clickedsubmit)
btnsubmit.grid(column=2,row=2)
btnregister = tkinter.Button(log,text="Register",bg="royal
blue",fg="white",font=("Times New Roman", 15),width=7,command=register)
btnregister.grid(column=2,row=1)

log.mainloop()

print(name)
```

```
global usrid
#Find user id from either user or critic table
if critic == 0:
      cursor.execute("SELECT usid FROM FPMovies.dbo.usr WHERE usname = \'" + 
name + "\'")
else:
      cursor.execute("SELECT cid FROM FPMovies.dbo.critic WHERE cname = \'" + 
name + "\'")
usrid = str(cursor.fetchone())
usrid = usrid.replace("(", "")
usrid = usrid.replace("'", "")
usrid = usrid.replace(")", "")
usrid = usrid.replace(",", "")
usrid = usrid.replace(" ", "")

#Rate - activated when a movie (from main window) is selected and then a 
rating (from movie review window) is clicked
def rate(stars):
      global load
      load = stars
      global list
      global usrid
      global name
      global movnum
      global critic

      #Find movie id
      numquery = "SELECT mid FROM FPMovies.dbo.movie WHERE mname = \'" + 
list.get('anchor') + "\'"
      cursor.execute(numquery)
      movnum = str(cursor.fetchone())
      movnum = movnum.replace("(", "")
      movnum = movnum.replace("'", "")
      movnum = movnum.replace(")", "")
      movnum = movnum.replace(",", "")
      movnum = movnum.replace(" ", "")

      #Update rating based on rating clicked and update the tables
      if(stars=="1" or stars=="2" or stars=="3" or stars=="4" or 
stars=="5"):
            if critic==0:
                  starquery = "UPDATE FPMovies.dbo.review SET review.r_score 
= \'" + stars + "\' WHERE r_uid = \'" + usrid + "\' AND r_mid = \'" + movnum 
+ "\'"
            else:
                  starquery = "UPDATE FPMovies.dbo.critic_review SET 
critic_review.c_score = \'" + stars + "\' WHERE c_cid = \'" + usrid + "\' AND 
c_mid = \'" + movnum + "\'"
            cursor.execute(starquery)
            conn.commit()

      #Update the button colors depending on if one has been clicked
      if(stars == "1"):
            movie_btn1.configure(bg="tomato")
            movie_btn2.configure(bg="ivory3")
            movie_btn3.configure(bg="ivory3",fg="white")
            movie_btn4.configure(bg="ivory3")
            movie_btn5.configure(bg="ivory3")
      elif(stars == "2"):
            movie_btn1.configure(bg="ivory3")
            movie_btn2.configure(bg="tan1")
            movie_btn3.configure(bg="ivory3",fg="white")
            movie_btn4.configure(bg="ivory3")
            movie_btn5.configure(bg="ivory3")
      elif(stars == "3"):
            movie_btn1.configure(bg="ivory3")
            movie_btn2.configure(bg="ivory3")
            movie_btn3.configure(bg="gold2",fg="black")
            movie_btn4.configure(bg="ivory3")
            movie_btn5.configure(bg="ivory3")
      elif(stars == "4"):
            movie_btn1.configure(bg="ivory3")
            movie_btn2.configure(bg="ivory3")
            movie_btn3.configure(bg="ivory3",fg="white")
            movie_btn4.configure(bg="SeaGreen3")
            movie_btn5.configure(bg="ivory3")
      elif(stars == "5"):
            movie_btn1.configure(bg="ivory3")
            movie_btn2.configure(bg="ivory3")
            movie_btn3.configure(bg="ivory3",fg="white")
            movie_btn4.configure(bg="ivory3")
            movie_btn5.configure(bg="SlateBlue1")
      else:
            print(stars)


#Display review text - activated when a movie is selected
def displayrvwtxt():
      global load
      global reviewtext
      global critic

      if(load != "None"):
            #Load previous review comments
            if critic == 0:
                  myrvw =
rvwtxt.find({'mid':movnum,'id':usrid,'type':'User'})
            else:
                  myrvw =
rvwtxt.find({'mid':movnum,'id':usrid,'type':'Critic'})

            for ld in myrvw:
                  ld = str(ld)
                  print(ld)
                  start=ld.find('\'review\': \'')
                  end=ld.rfind('}')
                  print(ld[start+11:end-1])
                  reviewtext.insert(tkinter.END,ld[start+11:end-1])


#Submit review - activated when submit button is clicked in review window

def submitreview():
      global mov
      global load
      global movnum
      global usrid
      global reviewtext
      global newreviewnum
      global critic

      #Display some movie information in console window
      print("MOVNUM = " + movnum)
      print("USRID = " + usrid)
      print("RVWTXT = " + reviewtext.get("1.0",'end-1c'))

      #Find review based on whether this is a critic or user reviewing the
movie
      if critic == 0:
            check = rvwtxt.find_one({'mid':movnum,'id':usrid,'type':'User'})
      else:
            check =
rvwtxt.find_one({'mid':movnum,'id':usrid,'type':'Critic'})
      print(check)

      #Update the movie review
      if("None" not in str(check)):
            if critic==0:
                  result =
rvwtxt.update_one({'mid':movnum,'id':usrid,'type':'User'},{'$set':
{'review':reviewtext.get("1.0",'end-1c')}})
            else:
                  result =
rvwtxt.update_one({'mid':movnum,'id':usrid,'type':'Critic'},{'$set':
{'review':reviewtext.get("1.0",'end-1c')}})
      else:
            if critic==0:

      myreview={'mid':movnum,'type':'User','id':usrid,'review':reviewtext.get
("1.0",'end-1c')}
            else:

      myreview={'mid':movnum,'type':'Critic','id':usrid,'review':reviewtext.g
et("1.0",'end-1c')}
            result = rvwtxt.insert_one(myreview)

      #Print the result in console window
      print("RESULT = " + str(result))
      mov.destroy()


#Window for movie review - activated by selecting a movie in main window
def movie():
      global mov
      mov = tkinter.Tk()
      global list
      global critic

      #Format tkinter window
      mov.title(list.get('anchor'))
      mov.geometry('425x400')
      #Window background color
      mov.configure(bg='ivory2')

      global newreviewnum
      global movnum

      #Find movie number
      numquery = "SELECT mid FROM FPMovies.dbo.movie WHERE mname = \'" +
list.get('anchor') + "\'"
      cursor.execute(numquery)
      movnum = str(cursor.fetchone())
      movnum = movnum.replace("(", "")
      movnum = movnum.replace("'", "")
      movnum = movnum.replace(")", "")
      movnum = movnum.replace(",", "")
      movnum = movnum.replace(" ", "")


      #Display movie cover image
      '''#Movie cover image
      #try:
      picquery = "SELECT imlink FROM FPMovies.dbo.movie WHERE mname = \'" +
list.get('anchor') + "\'"
      cursor.execute(picquery)
      url = str(cursor.fetchone())
      url = url.replace("(", "")
      url = url.replace("'", "")
      url = url.replace(")", "")
      url = url.replace(",", "")
      url = url.replace(" ", "")

      try:
            req = requests.get(url)
      except requests.exceptions.RequestException as e:
            print ("ERROR Making request")

      urllib.request.urlretrieve(url,"cover.jpg")

      try:
            image = Image.open('cover.jpg')
      except IOError:
            print("ERROR Can't open image")

      #canvas = tkinter.Canvas(mov,width=10,height=10,bg='white')
      cover = ImageTk.PhotoImage(image)
      #canvas.create_image((0,0),image=cover,state="normal",anchor=tkinter.NW
)
      #canvas.grid(row=0,column=0)
      label = tkinter.Label(mov,image=cover)
      label.image=cover
      #label.grid(row=0,column=0)
      label.place(x=0,y=0)'''

      #Find movie score
      if critic==0:
            initquery = "SELECT review.r_score FROM FPMovies.dbo.review INNER 
JOIN FPMovies.dbo.movie ON review.r_mid = movie.mid INNER JOIN 
FPMovies.dbo.usr ON review.r_uid = usr.usid WHERE usr.usname = \'" + name + 
"\' AND movie.mname = \'" + list.get('anchor') + "\'"
      else:
            initquery = "SELECT critic_review.c_score FROM 
FPMovies.dbo.critic_review INNER JOIN FPMovies.dbo.movie ON 
critic_review.c_mid = movie.mid INNER JOIN FPMovies.dbo.critic ON 
critic_review.c_cid = critic.cid WHERE critic.cname = \'" + name + "\' AND 
movie.mname = \'" + list.get('anchor') + "\'"
      cursor.execute(initquery)

      #Loaded movie score
      global load
      load = str(cursor.fetchone())
      load = load.replace("(", "")
      load = load.replace("'", "")
      load = load.replace(")", "")
      load = load.replace(",", "")
      load = load.replace(" ", "")
      print(load)

      temp = load

      '''Next, load the score and review data into the window'''
      movie_label1 = tkinter.Label(mov,text="My Review",font=("Times New 
Roman",15),bg="royal blue",fg="white")
      movie_label1.grid(column=0,row=0,columnspan=5,sticky=tkinter.W+tkinter.
E+tkinter.N+tkinter.S)

      movie_label2 = tkinter.Label(mov,text=list.get('anchor'),font=("Times 
New Roman",15),bg="royal blue",fg="white")
      movie_label2.grid(column=0,row=1,columnspan=5,sticky=tkinter.W+tkinter.
E+tkinter.N+tkinter.S)

      global mylistpalnums
      if critic == 0:
            #Friend Score
            scorequeryfriend = "SELECT ROUND(AVG(CAST(review.r_score AS 
FLOAT)),3) FROM FPMovies.dbo.review WHERE review.r_uid IN (" + 
str(mylistpalnums)[1:len(str(mylistpalnums))-1] + ") AND review.r_mid = \'" + 
movnum + "\'"
            cursor.execute(scorequeryfriend)
            score = str(cursor.fetchone())
            score = score.replace("(", "")
            score = score.replace("'", "")
            score = score.replace(")", "")
            score = score.replace(",", "")
            score = score.replace(" ", "")
            score = score.replace("\"","")
            friendscore = tkinter.Label(mov, text= "Friend Avg Score = " + 
score,font=("Times New Roman",14),bg = "royal blue",fg="white")

      friendscore.grid(column=0,row=2,columnspan=5,sticky=tkinter.W+tkinter.E
+tkinter.N+tkinter.S)

      #User Score
      scorequeryuser = "SELECT ROUND(AVG(CAST(review.r_score AS FLOAT)),3)
FROM FPMovies.dbo.review WHERE review.r_mid = \'" + movnum + "\'"
      cursor.execute(scorequeryuser)
      score = str(cursor.fetchone())
      score = score.replace("(", "")
      score = score.replace("'", "")
      score = score.replace(")", "")
      score = score.replace(",", "")
      score = score.replace(" ", "")
      score = score.replace("\"","")
      friendscore = tkinter.Label(mov, text= "User Avg Score = " +
score,font=("Times New Roman",14),bg = "royal blue",fg="white")
      friendscore.grid(column=0,row=3,columnspan=5,sticky=tkinter.W+tkinter.E
+tkinter.N+tkinter.S)

      #Critic Score
      scorequerycritic = "SELECT ROUND(AVG(CAST(critic_review.c_score AS
FLOAT)),3) FROM FPMovies.dbo.critic_review WHERE critic_review.c_mid = \'" +
movnum + "\'"
      cursor.execute(scorequerycritic)
      score = str(cursor.fetchone())
      score = score.replace("(", "")
      score = score.replace("'", "")
      score = score.replace(")", "")
      score = score.replace(",", "")
      score = score.replace(" ", "")
      score = score.replace("\"","")
      friendscore = tkinter.Label(mov, text= "Critic Avg Score = " +
score,font=("Times New Roman",14),bg = "royal blue",fg="white")
      friendscore.grid(column=0,row=4,columnspan=5,sticky=tkinter.W+tkinter.E
+tkinter.N+tkinter.S)

      #Movie rating buttons
      global movie_btn1
      global movie_btn2
      global movie_btn3
      global movie_btn4
      global movie_btn5
      movie_btn1=tkinter.Button(mov,text="1 Star
>:(",bg="ivory3",fg="white",font=("Times New Roman",14),command=lambda:
rate("1"))
      movie_btn1.grid(column=0,row=5)
      movie_btn2=tkinter.Button(mov,text="2 Stars
:(",bg="ivory3",fg="white",font=("Times New Roman",14),command=lambda:
rate("2"))
      movie_btn2.grid(column=1,row=5)
      movie_btn3=tkinter.Button(mov,text="3 Stars
:/",bg="ivory3",fg="white",font=("Times New Roman",14),command=lambda:
rate("3"))
      movie_btn3.grid(column=2,row=5)
      movie_btn4=tkinter.Button(mov,text="4 Stars
:)",bg="ivory3",fg="white",font=("Times New Roman",14),command=lambda:
rate("4"))
      movie_btn4.grid(column=3,row=5)
      movie_btn5=tkinter.Button(mov,text="5 Stars
:D",bg="ivory3",fg="white",font=("Times New Roman",14),command=lambda:
rate("5"))
```

```
	movie_btn5.grid(column=4,row=5)

	#Find review based on movie id, user id, and type of review
	global usrid
	global rvwtxt
	load = temp
	if critic == 0:
		check = rvwtxt.find_one({'mid':movnum,'id':usrid,'type':'User'});
	else:
		check =
rvwtxt.find_one({'mid':movnum,'id':usrid,'type':'Critic'});
	print(check)

	#If there is an existing review, launch the rating for that review
	if(load != "None" and check != "None"):
		rate(load)
		newreviewnum = 0
	elif(load != "None" and check == "None"):
		rate(load)
		newreviewnum = 1
	#Otherwise, make a new review and commit the changes to the database
	else:
		print("USRID = " + usrid)
		print("MOVNUM = " + movnum)
		if critic==0:
			newreview = "INSERT INTO FPMovies.dbo.review (r_mid, r_uid,
r_score) VALUES (\'" + movnum + "\',\'" + usrid + "\',\'" + str(1) + "\')"
		else:
			newreview = "INSERT INTO FPMovies.dbo.critic_review (c_mid,
c_cid, c_score) VALUES (\'" + movnum + "\',\'" + usrid + "\',\'" + str(1) +
"\')"
		cursor.execute(newreview)
		conn.commit()
		newreviewnum = 1

	#Set up review text entry and submit button
	global reviewtext
	reviewtext = tkinter.Text(mov,width=30,height=10)
	reviewtext.grid(column=0,row=7,columnspan=5,sticky=tkinter.W+tkinter.E+
tkinter.N+tkinter.S)

	btnsubmitrvw = tkinter.Button(mov,text="Submit",bg="royal
blue",fg="white",font=("Times New Roman",14),command=lambda: submitreview())
	btnsubmitrvw.grid(column=0,row=8,columnspan=5,sticky=tkinter.W+tkinter.
E+tkinter.N+tkinter.S)

	#If there is an existing review, load the review text
	if(load != "None" or check != "None"):
		displayrvwtxt()

	mov.mainloop()

#This closes the window if login failed and/or user clicked exit before
logging in
if auth == 0:
	sys.exit()
```

```
#Tkinter window
top = tkinter.Tk()

#Label the window
top.title("FP Movies")

#Window size
top.geometry('400x300')

#Window background color
top.configure(bg='ivory2')

menu = Menu(top)
menu.add_command(label = name)
top.config(menu=menu)

#Header label
lbl1 =tkinter.Label(top,text="FP Movies",font=("Times New
Roman",20),bg="ivory3",width = 10)
lbl1.grid(column=1,row=0)

#Search entry box
search=tkinter.Entry(top,width=20)
search.grid(column=1,row=1)

#Scrollbar
scroll = tkinter.Scrollbar(top)
scroll.grid(column=3,row=3,rowspan=30,sticky=tkinter.W+tkinter.E+tkinter.N+tk
inter.S)

#Listbox for Scrollbar
list = Listbox(top, yscrollcommand=scroll.set)
list.grid(column=0,columnspan=2,row=3,sticky=tkinter.W+tkinter.E+tkinter.N+tk
inter.S)

#Dropdown menu for search
combo = tkinter.ttk.Combobox(top,width=10)
combo['values'] = ("All","Movies","People","Settings")
combo.current(0)
combo.grid(column=0,row=1)

#Suggested movie listing which displays when movie button is clicked in main
window
def suggestedmovies():
      #Clear the list area in the window
      list.delete(0,'end')

      #Find the current user's friends
      pals = friends.find({'user':name})

      #Count the number of friends
      numfriends = friends.count_documents({'user':name})
      print("Num of Friends: " + str(numfriends))

      index1 = 0
      index2 = 0
      mylistpals = []
      sum = 0
      count = 0
      avg = 0.0

      #Add friend entries to a list of friends
      for ppl in pals:
            ppl = str(ppl)
            start=ppl.find('\'friend\': \'')
            end=ppl.rfind('}')
            print(ppl[start+11:end-2])
            mylistpals.append(ppl[start+11:end-2])
            print(mylistpals)

      #Convert those friend names in the list to a list of friend numbers
      global mylistpalnums
      mylistpalnums = []
      frquery = "SELECT usr.usid FROM FPMovies.dbo.usr WHERE usr.usname IN ("
+ str(mylistpals)[1:len(str(mylistpals))-1] + ")"
      print(frquery)
      cursor.execute(frquery)
      for fr in cursor:
            fr = str(fr)
            fr = fr.replace("(", "")
            fr = fr.replace("'", "")
            fr = fr.replace(")", "")
            fr = fr.replace(",", "")
            mylistpalnums.append(fr)

      #List the movies that have been reviewed by friends in order based on
the average rating for the movie
      avgquery1 = "SELECT movie.mname FROM FPMovies.dbo.review LEFT OUTER
JOIN FPMovies.dbo.movie ON review.r_mid = movie.mid WHERE review.r_uid IN ("
+ str(mylistpalnums)[1:len(str(mylistpalnums))-1] + ") GROUP BY movie.mname
ORDER BY ROUND(AVG(CAST(review.r_score AS FLOAT)),3) DESC"
      cursor.execute(avgquery1)

      #Display the movie name
      for res in cursor:
            res = str(res)
            res = res.replace("(", "")
            res = res.replace("'", "")
            res = res.replace(")", "")
            res = res.replace(",", "")
            list.insert(tkinter.END,res)
            print(res)

#List movies based on "srchm" criteria
def listmovies(srchm):
      #Clear the list area in the window
      list.delete(0,'end')

      #Specify the query based on search criteria
      if srchm == "All":
            query = "SELECT mname FROM FPMovies.dbo.Movie"
      else:
            query = "SELECT mname FROM FPMovies.dbo.movie where mname like
\'%" + srchm + "%\'"
```

```
      cursor.execute(query)

      #Display the movie name
      for row in cursor:
            row = str(row)
            row = row.replace("(", "")
            row = row.replace("'", "")
            row = row.replace(")", "")
            row = row.replace(",", "")
            list.insert(tkinter.END,row)
            print(row)
      list.grid(column=0,columnspan=3,row=3,sticky=tkinter.W+tkinter.E+tkinte
r.N+tkinter.S)
      scroll.config(command=list.yview)

#Add friend - activated by selecting a person and clicking "add friend"
def addfriend():
      global list
      newfriend = list.get('anchor')
      print("USER = " + name)
      print("FRIEND = " + str(newfriend))
      friendship={'user':name,'friend':newfriend}
      if newfriend != "" and newfriend != "None":
            fresult = friends.insert_one(friendship)
      print("FRESULT = " + str(fresult))
      clickedf()

#Remove friend - activated by selecting a friend and then clicking "remove
friend"
def removefriend():
      global list
      notfriend = list.get('anchor')
      print("USER = " + name)
      print("FRIEND = " + str(notfriend))
      unfriend={'user':name,'friend':notfriend}
      fresult = friends.delete_one(unfriend)
      print("FRESULT = " + str(fresult))
      clickedf()

#List people based on "srchp" criteria
def listpeople(srchp):
      #Clear the list area in the window
      list.delete(0,'end')

      #Specify the query based on the search criteria
      if srchp == "All":
            query = "SELECT usname FROM FPMovies.dbo.usr"
      else:
            query = "SELECT usname FROM FPMovies.dbo.usr where usname like
\'%" + srchp + "%\'"
      cursor.execute(query)

      #Display the results of the query
      for row in cursor:
            row = str(row)
            row = row.replace("(", "")
            row = row.replace("'", "")
            row = row.replace(")", "")
            row = row.replace(",", "")
            #mn = tkinter.Label(top,text=row,font=("Times New 
Roman",12),width=10)
            list.insert(tkinter.END,row)
            #txt1.insert(tkinter.END,mn)
            #txt1.insert(tkinter.END,"\n")
            print(row)
      list.grid(column=0,columnspan=3,row=3,sticky=tkinter.W+tkinter.E+tkinte
r.N+tkinter.S)
      scroll.config(command=list.yview)

#List friends based on "srchf" criteria
def listfriends(srchf):
      #Clear the list area in the window
      list.delete(0,'end')

      #Specify the query based on the search criteria
      if srchf == "All":
            #Find friends in NOSQL database
            pals = friends.find({'user':name})

            #Display results
            for ppl in pals:
                  ppl = str(ppl)
                  start=ppl.find('\'friend\': \'')
                  #print("START = " + str(start))
                  end=ppl.rfind('}')
                  #print("END = " + str(end))
                  print(ppl[start+11:end-1])
                  list.insert(tkinter.END,ppl[start+11:end-1])

#Action for when search button is clicked
def clickedsrch():
      global list
      global critic

      #Clear list area in the window, configure the window
      list.delete(0,'end')
      lbl1.configure(text="Search")
      txt1.delete('1.0',tkinter.END)

      #Retrieve the combobox selection and typed search information
      key = search.get()
      comb = combo.get()

      global button

      #Set up the select button based on the combobox selection
      if comb == "People":
            query = "SELECT usname FROM FPMovies.dbo.usr where usname like 
\'%" + key + "%\'"
            cursor.execute(query)
            if critic==0:
                  button = tkinter.Button(top,text="Add Friend",font=("Times 
New Roman",15),width=9,command=lambda x=list.get('anchor'): addfriend())
                  button.grid(column=2,row=3)
            for row in cursor:
                  row = str(row)
                  row = row.replace("(", "")
                  row = row.replace("'", "")
                  row = row.replace(")", "")
                  row = row.replace(",", "")
                  list.insert(tkinter.END,row)
                  txt1.insert(tkinter.END,row)
                  txt1.insert(tkinter.END,"\n")
                  print(row)
      elif comb == "Movies":
            listmovies(key)
            button = tkinter.Button(top,text="Select",font=("Times New 
Roman",15),width=9,command=lambda x=list.get('anchor'): movie())
            button.grid(column=2,row=3)

#Search button
btnsrch=tkinter.Button(top,text="Search",bg="royal 
blue",fg="white",font=("Times New Roman",15),width=7,command=clickedsrch)
btnsrch.grid(column=2,row=1)

#Clicked friends button
def clickedf():
      global list
      global critic
      list.delete(0,'end')
      lbl1.configure(text="Friends")
      txt1.delete('1.0',tkinter.END)
      combo.current(2)
      #listpeople("All")
      listfriends("All")
      global button
      if critic==0:
            button = tkinter.Button(top,text="Remove",font=("Times New 
Roman",15),width=9,command=lambda x=list.get('anchor'): removefriend())
            button.grid(column=2,row=3)

#Friends button
btnf=tkinter.Button(top,text="Friends",bg="ivory3",font=("Times New 
Roman",15),width=7,command=clickedf)
btnf.grid(column=0,row=10)

#Clicked movies button
def clickedm():
      global critic
      list.delete(0,'end')
      lbl1.configure(text="Movies")
      '''cursor.execute('SELECT mname FROM FPMovies.dbo.Movie')
      txt1.delete('1.0',tkinter.END)'''
      combo.current(1)
      all = "All"
      if critic==0:
            suggestedmovies()
      else:
            listmovies(all)
      button = tkinter.Button(top,text="Select",font=("Times New 
Roman",15),width=9,command=lambda x=list.get('anchor'): movie())
```

```
      button.grid(column=2,row=3)

#Movies button
btnm=tkinter.Button(top,text="Movies",bg="ivory3",font=("Times New
Roman",15),width=7,command=clickedm)
btnm.grid(column=1,row=10)

#Clicked settings button
def clickeds():
      list.delete(0,'end')
      lbl1.configure(text="Settings")
      txt1.delete('1.0',tkinter.END)
      combo.current(3)

#Settings button
btns=tkinter.Button(top,text="Settings",bg="ivory3",font=("Times New
Roman",15),width=7,command=clickeds)
btns.grid(column=2,row=10)

#Text box
txt1=scrolledtext.ScrolledText(top,height=20,width=10,yscrollcommand=scroll.s
et)
#txt1.grid(column=0,columnspan=3,row=3,sticky=tkinter.W+tkinter.E+tkinter.N+t
kinter.S)

top.mainloop()
```

FPMovies.sql

```
/*
Chris Collier, Vincent Crabtree, Daniel Kobold
ECE 59500 Database Management Systems
Final Project
*/

CREATE TABLE movie(
      mid integer NOT NULL PRIMARY KEY,
      mname varchar(40) NOT NULL,
      mdir varchar(20),
      myr integer,
      mgen varchar(15)
);

ALTER TABLE movie
ADD imlink varchar(500);

CREATE TABLE usr(
      usid integer NOT NULL PRIMARY KEY,
      usname varchar(20) NOT NULL
);

ALTER TABLE usr
ADD uspass varchar(20);

CREATE TABLE critic(
      cid integer NOT NULL PRIMARY KEY,
      cname varchar(20) NOT NULL,
      cemp varchar(15)
);

ALTER TABLE critic
ADD cpass varchar(20);

CREATE TABLE review(
      r_mid integer REFERENCES movie(mid),
      r_uid integer REFERENCES usr(usid),
      r_score integer NOT NULL,
      r_notes varchar(1000)
);

CREATE TABLE critic_review(
      c_mid integer REFERENCES movie(mid),
      c_cid integer REFERENCES critic(cid),
      c_score integer NOT NULL,
      c_notes varchar(1000)
);

INSERT INTO movie
      (mid, mname, mdir, myr, mgen)
values
      (1, 'Snow White and the Seven Dwarfs', 'David Hand', 1937, 'Fantasy'),
      (2, 'Pinocchio', 'Ben Sharpsteen', 1940, 'Fantasy'),
      (3, 'Fantasia', 'Samuel Armstrong', 1940, 'Anthology'),
      (4, 'Dumbo', 'Ben Sharpsteen', 1941, 'Fantasy'),
      (5, 'Bambi', 'David Hand', 1942, 'Fantasy');

INSERT INTO movie
      (mid, mname, mdir, myr, mgen)
values
      (6, 'Cinderella', 'Clyde Geronimi', 1950, 'Fantasy'),
      (7, 'Alice in Wonderland', 'Clyde Geronimi', 1951, 'Fantasy'),
      (8, 'Peter Pan', 'Clyde Geronimi', 1953, 'Fantasy'),
      (9, 'Lady and the Tramp', 'Clyde Geronimi', 1955, 'Adventure'),
      (10, 'Sleeping Beauty', 'Clyde Geronimi', 1959, 'Fantasy');

INSERT INTO movie
      (mid, mname, mdir, myr, mgen)
values
      (11, 'One Hundred and One Dalmatians', 'Wolfgang Reitherman', 1961,
'Adventure'),
      (12, 'The Sword in the Stone', 'Wolfgang Reitherman', 1963, 'Fantasy'),
      (13, 'The Jungle Book', 'Wolfgang Reitherman', 1967, 'Fantasy'),
      (14, 'The Aristocats', 'Wolfgang Reitherman', 1970, 'Comedy'),
      (15, 'Robin Hood', 'Wolfgang Reitherman', 1973, 'Comedy'),
      (16, 'The Many Adventures of Winnie the Pooh', 'John Lounsbery', 1977,
'Anthology'),
      (17, 'The Rescuers', 'Wolfgang Reitherman', 1977, 'Adventure'),
      (18, 'Petes Dragon', 'Don Chaffey', 1977, 'Fantasy'),
      (19, 'The Fox and the Hound', 'Ted Berman', 1981, 'Drama'),
      (20, 'Tron', 'Steven Lisberger', 1982, 'Sci-Fi');

INSERT INTO usr
```

```
      (usid, usname)
values
      (1, 'Chris Collier'),
      (2, 'Vincent Crabtree'),
      (3, 'Daniel Kobold');

UPDATE usr
SET uspass = 'ccollier'
WHERE usid = 1;

UPDATE usr
SET uspass = 'vcrabtree'
WHERE usid = 2;

UPDATE usr
SET uspass = 'dkobold'
WHERE usid = 3;

INSERT INTO review
      (r_mid, r_uid, r_score)
values
      (5, 3, 2);

INSERT INTO critic
      (cid,cname,cpass)
values
      (1,'Paul Strict','pstrict');

      /*
UPDATE movie
SET imlink = 'https://is4-
ssl.mzstatic.com/image/thumb/Video69/v4/37/a3/61/37a361bf-99d4-0283-65d6-
39bb0b66fce0/Snow-White-And-The-Seven-
Dwarfs_Walt_sig_Keystone_2000x3000.jpg/268x0w.jpg'
WHERE mid = 1;*/
```