Anime Recommendation System

Dhairya Ajitkumar Patel
Meng ECE, University of Waterloo
Student ID: 20906076
da32pate@uwaterloo.ca

Rumna Samanta
Meng ECE, University of Waterloo
Student ID: 20883387
rsamanta@uwaterloo.ca

***Abstract*—The rapid growth of Japan's animation industry has resulted in a plethora of anime films that have piqued the interest of diverse audiences. Each anime film has its own distinct personality that corresponds to a certain user's preferences. The recommendation system based on search results can help the client to find appropriate anime recommendation based on genre, anime score, etc. In our study we have made use of Kaggle dataset of anime which contains information of around 17,562 anime which are rated by different 325,772 users, our system will help client to fetch results based on querying the data and getting the results based on scores, genre, age rating etc. In addition to that, we have also made a provision for Admin User to add, modify and delete any record in case of any additions or changes are required in the database.**

***Index Terms*—Anime, Registered User, Non-Registered User, Admin User, Client, Server, Command Line Interface (CLI)**

I. Introduction

Numerous types of activities can now be done online thanks to technological advancements. With the advent of the internet in real-time, which captures every piece of information received, some information is increasingly being stored in cyberspace. As a result, a lot of information is speculative. Because the information may be unrelated to them, it makes it difficult for consumers or users to find their preferences. The goal of the recommendation system is to solve this issue. A recommendation system allows users to easily find relevant content without having to search manually.

Users can use the recommendation system to find products that might be useful. In terms of anime and manga, more and more versions with engaging stories and series quality are appearing. Users will have a hard time choose which movie to view first due to this issues.

To overcome the above mentioned issues, in this project we will create a client based CLI which will help client to search the anime based on several search criteria like name, genre, ratings, popularity etc. Furthermore, we are also helping the database user who maintains the database record to modify and add new anime information without loading the data repetitively. The implementation of project is done using the dataset from Kaggle. [1]

II. The Overview and Implementation

A. Client - Server Architecture

As shown in Fig.1, our design is based on fundamental concept of client and server side architecture. The client here is the local computer which in our scope is CLI application. The Server may be local or remote. The Server is MySQL server which has our anime database. The Client/or CLI sends in the request to the server to fetch some information from the server. The MySQL server in response sends in the required result to the clients CLI.

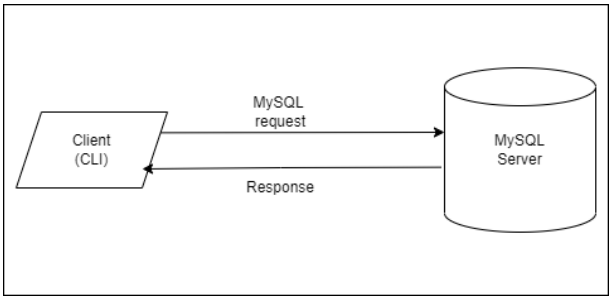

Fig. 1. Client - Server architecture

B. Implementation

The Project is based on three types of users:
1.) Non-Registered User
2.) Registered User
3.) Admin User

The Non-Registered User has the provision to only search anime based on name, anime age rating and date of release. The Registered User has an expanded provision to search results based on name, score, genre, anime popularity, age rating, synopsis. Besides this the Registered User can also modify his/her rating which he/she previously rated. The Admin User is the database(DB) admin who has the provision to add new anime also modify or update a particular anime record. In our database we have two admins who have the privilege to perform these operations.

In the following section we will describe our server side implementation i.e. the ER model in section C and ER model to relational schema in section D

C. ER model

In this section the Fig.2 describes the ER model of our Anime recommendation system. It describes the entities, their relationships and cardinality constraints. From the Fig. 2 Entity AnimeRatingComplete is in total participation with AnimeUserRatingWatchStatus. AnimeRatingComplete is inherited to AnimeUserWatchStatus For this project we have

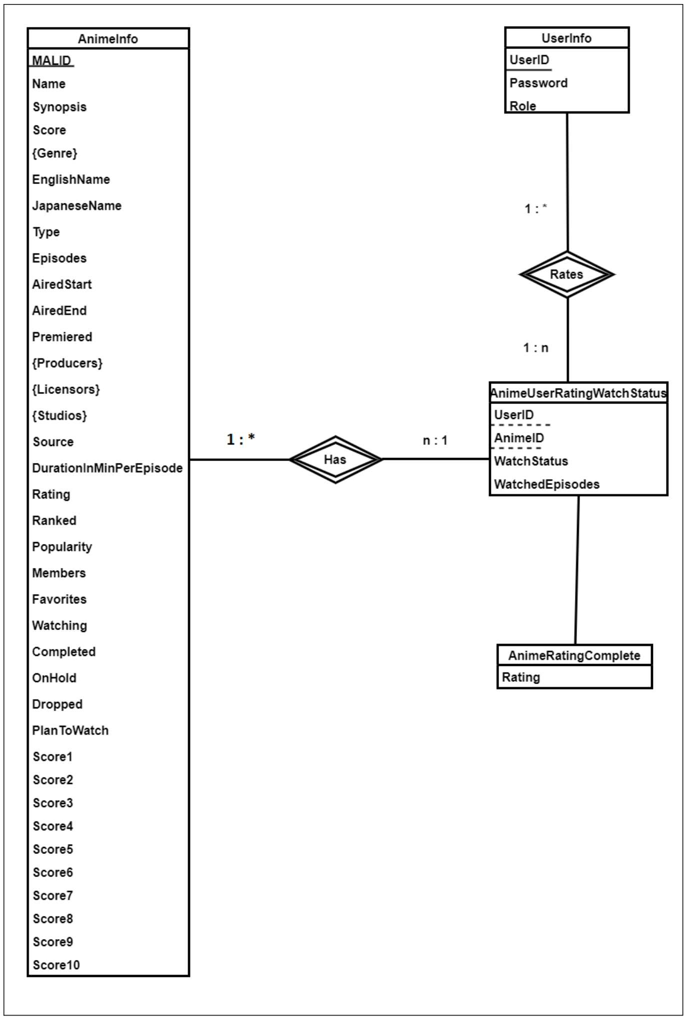

Fig. 2. AnimeInfo table

explicitly created UserInfo entity which has information of Client's ID, password and Role of the Client. There are two DB admins in our database and 353,402 Registered Users whose rating data are available.

D. ER to Relational Schema

We have translated our ER model to relational schema using appropriate foreign keys and primary keys of the tables as mentioned in the codes below. In AnimeInfo entity we had Producers, Licensors, Studios, Genres as multi-valued attribute which are separated into individual tables pre-processed using python scripts. The necessary data cleaning for fetching the aired start and end date of anime is done using MySQL. We have also cleaned the data for average time duration of anime episode and converted it into minutes. The necessary check constraints are added for the tables as shown in Fig 3 and Fig 6. We have put indexes on Genre in Genre table and Aired_start in AnimeInfo table.The detailed description of Foreign keys(FK) is as follows: AnimeUserRatingWatchStatus.UserID is FK to UserInfo.UserID, AnimeUserRatingWatchStatus.AnimeID is FK to AnimeInfo.MALID, AnimeRatingComplete.UserID and AnimeRatingComplete.MALID is FK to AnimeUserRatingWatchStatus.UserID and AnimeUserRatingWatchStatus.AnimeID respectively.

For Non-Registered user, instead of giving full access to tables we have created a view which has a portion of attributes from AnimeInfo table.

CREATE VIEW Anime_View_NonRegUser AS (select Name, JapaneseName, Rating, Score, EnglishName, Episodes, Aired_start, Duration_in_min_per_episode from animeinfo);

```sql
create database animeDB; use animeDB;
drop table if exists AnimeInfo;
create table AnimeInfo(MALID int not null , Name varchar(120) not null,Score float, EnglishName varchar(120),
    JapaneseName varchar(100), Type char(8), Episodes int, Aired varchar(30), Aired_start date, Aired_end date,
    Premiered VARCHAR(15), Source char(20), Duration VARCHAR(30),  hr_duration INT, min_duration INT,
    Duration_in_min_per_episode INT, Rating Char(10), Ranked int, Popularity int, Members int, Favourites int,
    Watching int, Completed int, OnHold int, Dropped int, PlanToWatch int,Score10 int,Score9 int, Score8 int,
    Score7 int,Score6 int,Score5 int,Score4 int,Score3 int,Score2 int,Score1 int,Synopsis longtext,
    PRIMARY KEY (MALID));
-- index on Aired_start
CREATE INDEX idx_duration ON AnimeInfo(Aired_start);
load data local infile 'C:/Fall21/ECE656/project_dataset/anime1.csv' ignore into table AnimeInfo
fields terminated by ','
enclosed by '"'
lines terminated by '\r\n'
ignore 1 lines
```

```sql
(@col1, @col2,@col3, @col4,@col5, @col6,@col7, @col8,@col9, @col10,@col11, @col12,@col13, @col14,@col15, @col16,
    @col17, @col18,@col19,@col20,@col21, @col22, @col23, @col24,@col25, @col26,@col27, @col28,@col29, @col30,
    @col31, @col32, @col33, @col34, @col35,@col36)
 set MALID = @col1, Name = @col2, Score = if(@col3 = 'Unknown', NULL, @col3), EnglishName = @col5,
    JapaneseName = @col6, Type = @col7, Episodes = if(@col8 = 'Unknown', NULL, @col8), Aired = @col9,
    Aired_start = NULL, Aired_end = Null, Premiered = @col10, Source= @col14, Duration = @col15,
    -- converting null to 0 and then converting from hrs to mins
    hr_duration = regexp_replace(regexp_substr(Duration, '^[0-9]{1,2} hr'),' hr',''),
    min_duration = regexp_replace(regexp_substr(@col15,'[0-9]{1,2} min'),' min',''),
    Duration_in_min_per_episode = (coalesce(hr_duration*60,0) + coalesce(min_duration,0)),
    Rating = regexp_substr(@col16,'(^R - 17\+)|(^PG-13)|(^PG)|(^R\+)|(G)|(^Rx)|(^Unknown)'),
    Ranked= if(@col17 = 'Unknown', NULL, @col17),Popularity=@col18, Members= @col19, Favourites = @col20,
    Watching = @col21, Completed= @col22,OnHold= @col23, Dropped= @col24, PlanToWatch= @col25,
    Score10= if(@col26 = 'Unknown', NULL, @col26), Score9=if(@col27 = 'Unknown', NULL, @col27),
    Score8= if(@col28 = 'Unknown', NULL, @col28), Score7 = if(@col29 = 'Unknown', NULL, @col29),
    Score6= if(@col30 = 'Unknown', NULL, @col30), Score5 = if(@col31 = 'Unknown', NULL, @col31),
```

```sql
    Score4= if(@col32 = 'Unknown', NULL, @col32), Score3 = if(@col33 = 'Unknown', NULL, @col33),
    Score2= if(@col34 = 'Unknown', NULL, @col34), Score1= if(@col35 = 'Unknown', NULL, @col35),Synopsis = @col36;
update AnimeInfo set Aired_start = str_to_date(regexp_replace(regexp_substr -- converting into date format
        (Aired, '^[a-zA-Z]{3} [0-9]{1,2}, [0-9]{4}'),',' ,''),"%M %d %Y");
update AnimeInfo set Aired_end = str_to_date(regexp_replace(regexp_substr(regexp_substr
        (Aired, 'to [a-zA-Z]{3} [0-9]{1,2}, [0-9]{4}$'),'[a-zA-Z]{3} [0-9]{1,2}, [0-9]{4}$'),',',''),"%M %d %Y"
update AnimeInfo set hr_duration = 0 where hr_duration is NULL;
update AnimeInfo set Rating = NULL where Rating ='Unknown';
update AnimeInfo set Premiered = NULL where Premiered ='Unknown';
update AnimeInfo set Source  = NULL where Source ='Unknown';
update AnimeInfo set Duration_in_min_per_episode = NULL where Duration_in_min_per_episode=0;
Alter table AnimeInfo drop Aired,drop Duration,drop hr_duration, drop min_duration;
ALTER TABLE animeinfo ADD CONSTRAINT CHK_Score CHECK (Score>=0 and Score<=10);
ALTER TABLE animeinfo ADD CONSTRAINT CHK_Episodes CHECK (Episodes>0);
ALTER TABLE animeinfo ADD CONSTRAINT CHK_Duration CHECK (Duration_in_min_per_episode>0);
ALTER TABLE animeinfo ADD CONSTRAINT CHK_MALID CHECK (MALID>0);
```

Fig. 3. AnimeInfo table

```sql
drop table if exists Genre;
CREATE TABLE Genre(MALID int not null, Genre char(30), Primary key (MALID,Genre),
                CONSTRAINT fk_Genre_AnimeInfo Foreign key (MALID) references animeinfo(MALID)on delete CASCADE);
CREATE INDEX idx_Genre ON Genre(Genre);
load data local infile 'C:/Fall21/ECE656/project_dataset/Genre.csv' ignore into table Genre
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
ignore 1 lines
(@col1,@col2)
set MALID = @col1, Genre = @col2;
-- removing tab and \n if any
UPDATE genre SET `Genre` = TRIM(REPLACE(REPLACE(REPLACE(`Genre`, '\n', ' '), '\r', ' '), '\t', ' '));
-- Producers>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
drop table if exists Producer;
CREATE TABLE Producer(MALID int not null, Producer char(50), Primary key (MALID,Producer),
              CONSTRAINT fk_Producer_AnimeInfo Foreign key (MALID) references animeinfo(MALID)on delete CASCADE);
```

E. Client Application

We have designed a traditional client based Command Line Interface that is used to fetch the results that Client chooses

```sql
load data local infile 'C:/Fall21/ECE656/project_dataset/Producers.csv' ignore into table Producer
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
ignore 1 lines
(@col1,@col2)
set MALID = @col1, Producer = @col2;
-- removing tab and \n if any
UPDATE Producer SET `Producer` = TRIM(REPLACE(REPLACE(REPLACE(`Producer`, '\n', ' '), '\r', ' '), '\t', ' '));
-- Licensors>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
drop table if exists Licensor;
CREATE TABLE Licensor(MALID int not null, Licensor char(50), Primary key (MALID,Licensor),
                CONSTRAINT fk_Licensor_AnimeInfo Foreign key (MALID) references animeinfo(MALID)on delete CASCADE);
load data local infile 'C:/Fall21/ECE656/project_dataset/Licensors.csv' ignore into table Licensor
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
ignore 1 lines
```

```sql
(@col1,@col2)
set MALID = @col1, Licensor = @col2;
-- removing tab and \n if any
UPDATE Licensor SET `Licensor` = TRIM(REPLACE(REPLACE(REPLACE(`Licensor`, '\n', ' '), '\r', ' '), '\t', ' '));
-- Studios>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
drop table if exists Studio;
CREATE TABLE Studio(MALID int not null, Studio char(50), Primary key (MALID,Studio),
                CONSTRAINT fk_Studio_AnimeInfo Foreign key (MALID) references animeinfo(MALID)on delete CASCADE);
load data local infile 'C:/Fall21/ECE656/project_dataset/Studios.csv' ignore into table Studio
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
ignore 1 lines
(@col1,@col2)
set MALID = @col1, Studio = @col2;
-- removing tab and \n if any
UPDATE Studio SET `Studio` = TRIM(REPLACE(REPLACE(REPLACE(`Studio`, '\n', ' '), '\r', ' '), '\t', ' '));
```

```sql
-- UserInfo>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
drop table if exists UserInfo;
CREATE TABLE UserInfo(UserID int not null, Password int not null, Role char(25), Primary key (UserID));
load data local infile 'C:/Fall21/ECE656/project_dataset/UserInfo.csv' ignore into table UserInfo
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
ignore 1 lines
(@col1,@col2,@col3)
set UserID = @col1, Password = @col2, Role = @col3;
-- removing tab and \n if any
UPDATE UserInfo SET `Role` = TRIM(REPLACE(REPLACE(REPLACE(`Role`, '\n', ' '), '\r', ' '), '\t', ' '));
```

Fig. 4. Multivalued attributes table

```sql
drop table if exists AnimeUserRatingWatchStatus;
create table AnimeUserRatingWatchStatus(UserID int not null,MALID int,Rating int, WatchingStatus INT,
        WatchedEpisodes int);
load data local infile 'C:/Fall21/ECE656/project_dataset/animelist1.csv' ignore into table
    AnimeUserRatingWatchStatus
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
ignore 1 lines
(UserID,MALID,Rating,WatchingStatus,WatchedEpisodes);
ALTER TABLE animeuserratingwatchstatus ADD CONSTRAINT
        PK_animeuserratingwatchstatus PRIMARY KEY (UserID,MALID);
ALTER TABLE animeuserratingwatchstatus ADD CONSTRAINT
        FK_watchstatus_userinfo FOREIGN KEY (UserID) REFERENCES userinfo(UserID)on delete cascade;
ALTER TABLE animeuserratingwatchstatus ADD CONSTRAINT
        FK_watchstatus_animeinfo FOREIGN KEY (MALID) REFERENCES animeinfo(MALID)on delete cascade;
```

Fig. 5. AnimeUserRatingWatchStatus table

```sql
drop table if exists AnimeRatingComplete;
create table AnimeRatingComplete(UserID int not null,MALID int not null,rating int not NULL);
load data local infile 'C:/Fall21/ECE656/project_dataset/rating_complete.csv' ignore
    into table AnimeRatingComplete
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
ignore 1 lines
(UserID,MALID,Rating);
ALTER TABLE AnimeRatingComplete ADD CONSTRAINT PK_AnimeRatingComplete PRIMARY KEY (UserID,MALID);
ALTER TABLE animeratingcomplete ADD CONSTRAINT FK_animeratingcomplete_watchstatus
        FOREIGN KEY (UserID) REFERENCES animeuserratingwatchstatus(UserID)on delete cascade;
ALTER TABLE animeratingcomplete ADD CONSTRAINT FK_animeratingcomplete_watchstatus1
        FOREIGN KEY (MALID) REFERENCES animeuserratingwatchstatus(MALID)on delete cascade;
ALTER TABLE animeratingcomplete ADD CONSTRAINT CHK_rating CHECK (rating>=0 and rating<=10);
```

Fig. 6. AnimeRatingComplete table

the options from the list of available provisions. The Client side coding is done using Python 3.9.

Non-Registered User has three functionalities i.e. to search anime based on name, age rating and release date. The client side coding is shown in Fig 7, Fig 8, Fig 9.

```python
# Code to search Animes based on Anime Name- Non-registered user
def SearchByAnimeNameNonRegUser():
    os.system('cls')
    AnimeName = input("Please type anime name to search: ")
    #Query string to fetch results
    UserAnimeQuery = "Select Name,Score,Episodes,Aired_start from Anime_View_NonRegUser where Name like \'" + AnimeName + "%\';"
    try:
        mycursor=mydb.cursor()
        mycursor.execute(UserAnimeQuery)
        animes = mycursor.fetchall()
        ColumnName=["Name", "Score","Episodes","Released Date"]
        display = PrettyTable()
        #If no record found
        if (mycursor.rowcount == 0):
            print("Oops! No record found...")
            ch = int(input("Press 0 to exit: "))
            if (ch == 0):
                return 0
```

Fig. 7. Search by name - NonRegistered User

```python
def SearchByAnimeRatingNonRegUser():
    os.system('cls')
    #Fetch Age ratings available in the DB
    UserAnimeQuery = "Select DISTINCT Rating from Anime_View_NonRegUser where Rating is not NULL;"
    mycursor = mydb.cursor(buffered=True)
    mycursor.execute(UserAnimeQuery)
    animes = mycursor.fetchall()
    items = tuple(list(list(zip(*animes))[0]))
    for a in range(0,len(items),1):
        print("Press ", a+1," to search animes with rating ",items[a])
    UserChoice=int(input("Please select one option: "))
    ch=items[UserChoice-1]
    #Code to fetch Animes based on Age rating selected by the user
    AnimeByRatingQuery="Select Name,Rating from Anime_View_NonRegUser where Rating=\'"\
                +str(ch)+"\';"
    print(AnimeByRatingQuery)
    try:
        mycursor.execute(AnimeByRatingQuery)
        results=mycursor.fetchall()
        ColumnName=["Anime Name","Rating"]
        display = PrettyTable()
        #If no record found
        if (mycursor.rowcount == 0):
            print("Oops! No record found...")
            ch = int(input("Press 0 to exit: "))
            if (ch == 0):
                return 0
```

Fig. 8. Search by age Rating - NonRegistered User

```python
#Code to search Animes based on relase Date-Non Registered
def SearchByAnimeReleaseDateNonRegUser():
    os.system('cls')
    AiredStart = input("Please type the 1st relase date of anime in (YYYY-MM-DD) format: ")
    while(1):
        if(AiredStart!=""):
            break
        else:
            os.system('cls')
            AiredStart = input("Please enter at least the 1st relase date of anime in (YYYY-MM-DD) format: ")
    AiredEnd =input("Please type the 2nd relase date of anime in (YYYY-MM-DD) format: ")
    if(AiredEnd!=""):
        SearchByDateQuery='Select Name,Aired_start from Anime_View_NonRegUser where Aired_start BETWEEN \''+AiredStart+'\' and \''+AiredEnd+'\';'
    else:
        SearchByDateQuery = 'Select Name,Aired_start from Anime_View_NonRegUser where Aired_start>=\'' + AiredStart + '\';'
    try:
        mycursor = mydb.cursor()
        mycursor.execute(SearchByDateQuery)
        animes = mycursor.fetchall()
        ColumnName=["Name", "Aired"]
        display = PrettyTable()
        if (mycursor.rowcount == 0):
            print("Oops! No record found...")
            ch = int(input("Press 0 to exit: "))
            if (ch == 0):
                return 0
```

Fig. 9. Search by release date - NonRegistered User

As mentioned Registered User has additional functionality of searching anime based on anime score which is shown in Fig 10. Fig. 11 shows the function to search based on Genre, having index on Genre makes the query run more efficiently. We have made a functionality where the user can see top trending anime. We have taken an input from user on how

```python
#Code to search Animes based on score- Registered User
def SearchByAnimeUserRating():
    os.system('cls')
    print("Press 1 see the animes with Score between 9-10")
    print("Press 2 see the animes with Score between 8-9")
    print("Press 3 see the animes with Score between 7-8")
    print("Press 4 see the animes with Score between 6-7")
    print("Press 5 see the animes with Score between 5-6")
    print("Press 0 to exit")
    UserChoice = int(input("Please enter your choice: "))
    if(UserChoice==1):
        SearchByAnimeRatingQuery="Select Name,Score from AnimeInfo where Score>=9 and Score<=10;"
    elif (UserChoice == 2):
        SearchByAnimeRatingQuery = "Select Name,Score from AnimeInfo where Score>=8 and Score<=9;"
    elif (UserChoice == 3):
        SearchByAnimeRatingQuery = "Select Name,Score from AnimeInfo where Score>=7 and Score<=8;"
    elif (UserChoice == 4):
        SearchByAnimeRatingQuery = "Select Name,Score from AnimeInfo where Score>=6 and Score<=7;"
    elif (UserChoice == 5):
        SearchByAnimeRatingQuery = "Select Name,Score from AnimeInfo where Score>=5 and Score<=6;"
    try:...

    except mysql.connector.IntegrityError as ie:
        print("Failed to fetch values...!!",ie)
```

Fig. 10. Search by anime scores - Registered User

```python
#Code to search Animes based on Genres- Registered User
def SearchByAnimeByGenre():
    #Code to fetch different Genres available in the DB
    GenreList="Select distinct Genre from Genre;"
    try:...
    except mysql.connector.IntegrityError as ie:
        print("Failed to fetch values...!!",ie)
        ch = int(input("Press 0 to exit: "))
        if (ch == 0):
            return 0
    #Code to prepare a Genre Name list seperated by ,
    GenreInput=input("Enter the genres seperated by , to search anime:")
    G_list=GenreInput.split(",")
    tempG=""
    if(len(G_list)>1):...
    else:
        tempG=str(G_list[0])
    #Code to fecth animes based on user Genre input
    if(len(G_list)==1):
        GenreQuery="SELECT Distinct Name from AnimeInfo INNER join Genre on AnimeInfo.MALID=Genre.MALID"\
                   " where Genre =\'"+str(tempG)+"\';"
    else:
        GenreQuery = "SELECT Distinct Name from AnimeInfo INNER join Genre on AnimeInfo.MALID=Genre.MALID"\
                     " where Genre =\'" + str(tempG) + ";"
```

Fig. 11. Search by Anime Genre - Registered User

many top trending anime he/she wants for eg. top 3 or top 5 trending anime. This functionality is shown in Fig 12 Figure

```python
#Code to seach Anime based on popularity - Registered User
def SearchForTrendingAnime():
    os.system('cls')
    limit=input("Please enter how many trending Anime you want to view: ")
    TrendingAnimeQuery="Select DISTINCT Name from AnimeInfo order by popularity DESC limit "+str(limit)+";"
    try:...
    except mysql.connector.IntegrityError as ie:
        print("Failed to fetch values...!!", ie)
        ch = int(input("Press 0 to exit: "))
        if (ch == 0):
            return 0
```

Fig. 12. Search top trending Anime - Registered User

14 shows the code foe how the registered user can modify his/her previously rated anime. The user can modify the rating and search again to view the updated results.

Admin user is a DB admin who can is able to insert/update an anime record, insert/delete a user from the database. Fig 15 and 16 shows the code for insertion into AnimeInfo and UserInfo table. Fig 17 shows the code for modifying anime record and Fig 18 shows the code for deleting a user record from Userinfo table.

```python
#Code to fetch synopsis of an Anime - Registered User
def FetchSynopsis():
    os.system('cls')
    AnimeName = input("Please type anime name to search: ")
    UserAnimeQuery = 'Select Name,Synopsis from AnimeInfo where Name like \'' + AnimeName + '%\';'
    try:...
    except mysql.connector.IntegrityError as ie:
        print("Failed to fetch values...!!", ie)
        ch = int(input("Press 0 to exit"))

        if (ch == 0):
            return 0
```

Fig. 13. Get anime synopsis - Registered User

```python
#Code to modify the rating of an Anime that the user had watched and rated previously - Registered User
def RateAnime(UserID):
    os.system('cls')
    print("List of Animes that you have watched and rated..")
    AnimeRatingQuery="Select AnimeInfo.Name, AnimeRatingComplete.Rating,AnimeRatingComplete.MALID from " \
                     "AnimeInfo INNER JOIN AnimeRatingComplete"\
                         " on AnimeInfo.MALID=AnimeRatingComplete.MALID where UserID="+str(UserID)+";"
    try:...
    except mysql.connector.IntegrityError as ie:
        print("Failed to fetch values...!!", ie)
        ch = int(input("Press 0 to exit: "))
        if (ch == 0):
            return 0
    AnimeName = input("Enter the anime name to modify the rating: ")
    UserRating = int(input("Enter a score between 0-10: "))
    for(Name,Rating,MALID) in animes:
        if(AnimeName==Name):
            UpdateRatingQuery="Update AnimeRatingComplete set Rating="+str(UserRating)+" where"\
                              " MALID="+str(MALID)+" and UserID="+str(UserID)+";"
            try:...
            except mysql.connector.IntegrityError as ie:
                print("Failed to update record...!!",ie)
                ch = int(input("Press 0 to exit: "))
                if (ch == 0):
                    return 0
```

Fig. 14. Modifying user's anime score - Registered User

```python
#Code for Admin to insert values into sql tables - Admin User
def InsertRecords():
    os.system('cls')
    print("Press 1 to insert into the table AnimeInfo")
    print("Press 2 to insert into the table UserInfo")
    print("Press 0 to exit")
    UserChoice=int(input("Please enter your choice: "))
    if(UserChoice==0):
        return 0
    elif(UserChoice==1):
        MALID = input("Input MALID: ")
        while (1):...
        Name=input("Input Anime Name: ")
        while(1):...
        EnglishName = input("Input Anime English Name: ")
        Type = input("Input Anime Type: ")
        Episodes = input("Input Anime Episodes: ")
        while (1):...
        format = "%Y-%m-%d"
        AiredStr=input("Input Anime Aired_start: ")
        while(1):...
        AiredEndStr = input("Input Anime Aired_end: ")
        while (1):...
        Duration_in_min_per_episode = input("Input Duration_in_min_per_episode: ")
        while (1):...
        Popularity = input("Input Popularity: ")
```

Fig. 15. Inserting a record - Admin User

```
    elif(UserChoice==2): #Insert into userinfo table - Admin user
        flag=0
        os.system('cls')
        while(1):...
        if (flag==0):
            lastrow="SELECT * FROM userinfo WHERE userid=(SELECT max(userid) FROM userinfo);"
            mycursor = mydb.cursor()
            mycursor.execute(lastrow)
            lastentry=mycursor.fetchall()
            for l in lastentry:
                lastUid=int(l[0])
            UserInfoInsertQuery="Insert into UserInfo " \
                                "values ("+str(lastUid+1)+", "+str(lastUid+1)+", \'"+role+"\');"
            try:...
            except mysql.connector.IntegrityError as ie:...

            except mysql.connector.errors.DatabaseError as er:...
```

Fig. 16. Inserting a record - Admin User

```
#Code for Admin to update records in sql table - Admin User
def UpdateRecords():
    os.system('cls')
    while(1):
        MALID=int(input("Enter MALID to update record in AnimeInfo table: "))
        if (isinstance(MALID, int)):
            break
        else:
            MALID = input("Wrong input! Please enter a valid MALID: ")
    AttributeName=input("Enter the Name of the attribute to be updated: ")
    AttributeValue=input("Enter the value of the attribute to be updated: ")
    SearchQuery="Select * from animeinfo where MALID="+str(MALID)+";"
    try:
        mycursor=mydb.cursor(buffered=True)
        mycursor.execute(SearchQuery)
        if(mycursor.rowcount==0):...
        else:
            UpdateQuery="Update AnimeInfo set "+str(AttributeName)+"=\'"+str(AttributeValue)+"\' " \
                        "where MALID="+str(MALID)+";"
            try:...
            except mysql.connector.IntegrityError as ie:...
            except mysql.connector.errors.DatabaseError as er:...
    except mysql.connector.IntegrityError as ie:...0
    except mysql.connector.errors.DatabaseError as er:...
```

Fig. 17. Updating anime record - Admin User

```
#Code for Admin to delete a record from sql table- Admin User
def DeleteRecord():
    os.system('cls')
    uid = int(input("Enter UserID to delete a record from the UserInfo table: "))
    UserDeleteQuery = "Delete from UserInfo where UserID=" + str(uid) + ";"
    try:
        mycursor = mydb.cursor()
        mycursor.execute(UserDeleteQuery)
        mydb.commit()
        print("Record deleted successfully..")
        mycursor.close()
        ch = int(input("Press 0 to exit: "))
        if (ch == 0):
            return 0
    except mysql.connector.IntegrityError as ie:
        print("Failed to insert values...!!",ie)
        ch = int(input("Press 0 to exit: "))
        if (ch == 0):
            return 0
    except mysql.connector.errors.DatabaseError as er:
        print("Error: {}".format(er))
        ch = int(input("Press 0 to exit: "))
        if (ch == 0):
            return 0
```

Fig. 18. Delete user record - Admin User

III. Testing the cases in CLI

Now that we have seen the codes which will be used for testing our recommendation system, we will now show and run some test cases and validate the results on the expected output. We have run command prompt with administrator rights and changed the location to the directory where all the .py files are present (here its in *C:\users\dhair*). The main file is named as AnimeMain.py

A. Testing for Non-Registered Users

Fig 19 shows the main screen where users are allowed to select from the available options. Non-Registered Users are able to choose option 2.

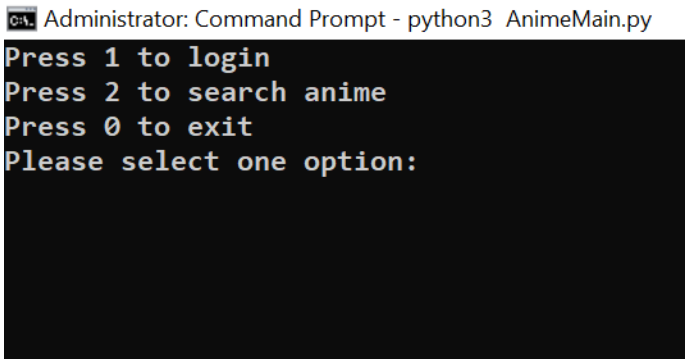

Fig. 19. Home Screen

On selecting option 2 the user is now able to get the anime results based on Name, Anime Age Rating and Release Date (Fig 20).

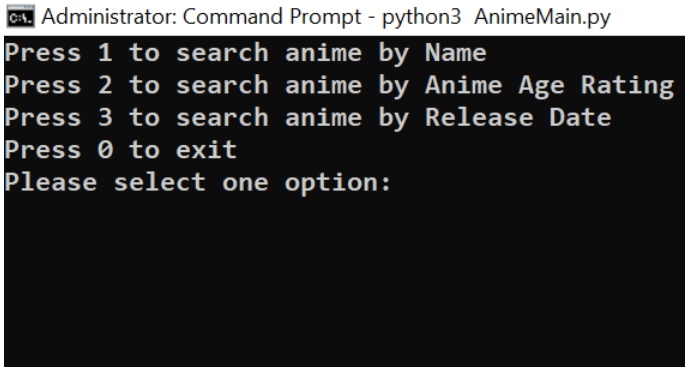

Fig. 20. Options for Non Registered Users

The non-registered user searches for ”Cowboy Bebop”. The results are shown in Fig.21

Now if the user chooses option ’0’ and chooses to search the anime based on release dates he will be prompted to enter two dates. The result displayed will list the anime who were premiered between the two dates. The results are shown in fig.22

Presence of index on Aired_start optimises searching for results. Figure 23 shows the explain command that is run on

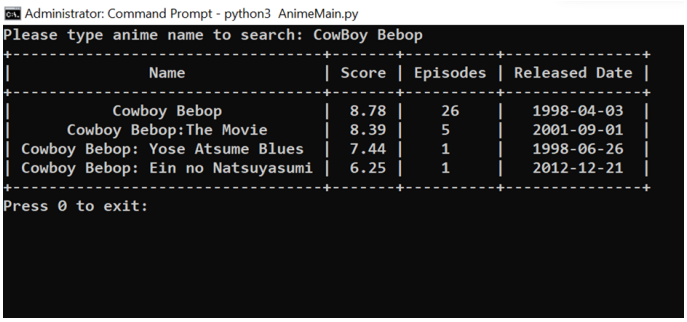

Fig. 21. Search results based on Name

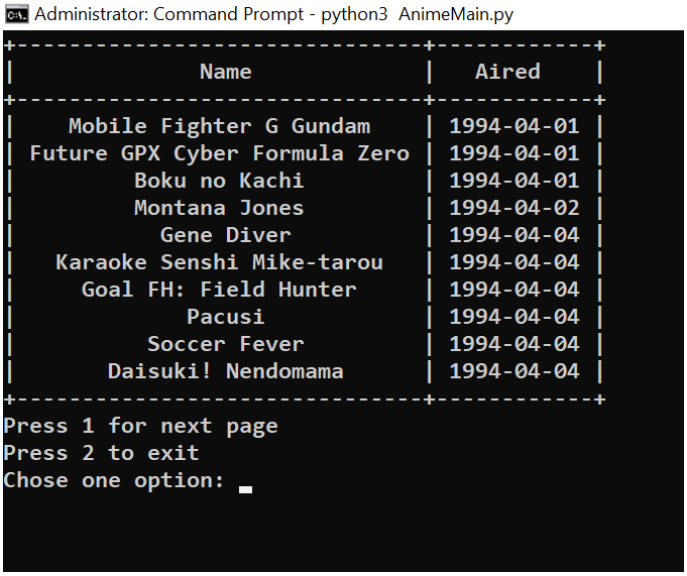

Fig. 22. Search results based on Release Date of Anime

MySQL Workbench which confirms the use of index named idx_duration as initialized in DDL statements (Fig 3).

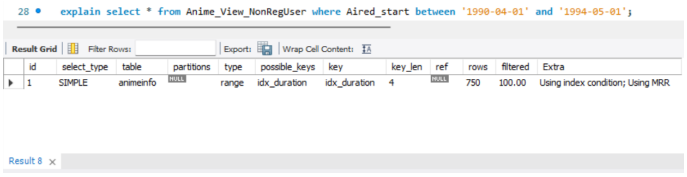

Fig. 23. 'EXPLAIN' command on search query based on release date

B. Testing for Registered Users

If the user is Registered user then the user will be asked to enter the his/her ID and password after choosing option 1 of the main screen. The home screen for Registered User after logging in is as shown in Fig. 24. The user has option to choose the Anime based on Name, Scores, Genre, Trending anime, search for synopsis of a particular anime and modify his/her anime score which the user previously rated.

For searching the results based on Genre, we have index idx_Genre. The search results for genre 'Action' is shown in

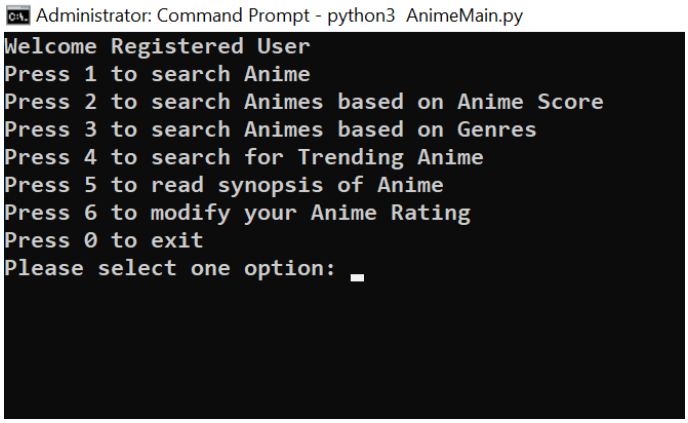

Fig. 24. Options for Registered User

Fig. 25. The explain command is shown in Fig.26 confirms the use of created index.

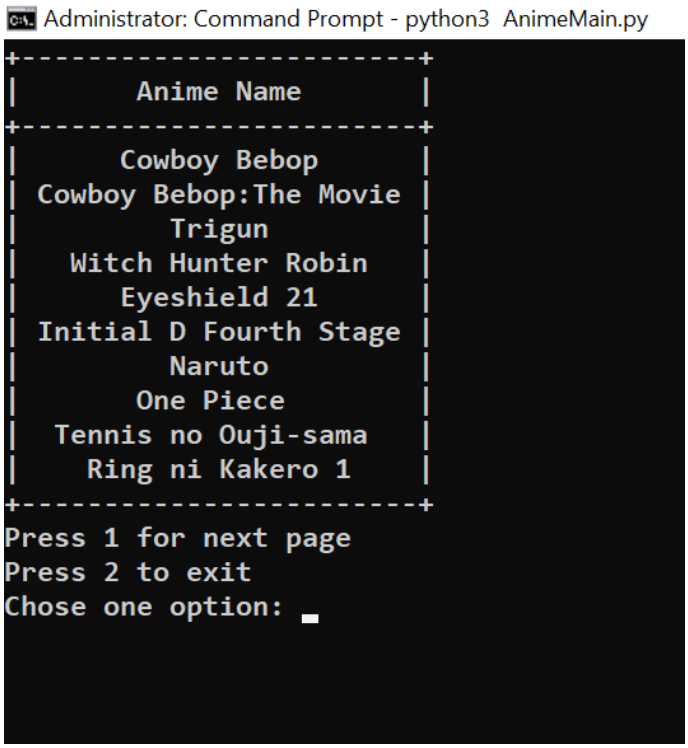

Fig. 25. Results based on Genre

Recommendation of Anime can be helpful to user based on popularity. User might want to get a list most popular anime which are trending. We have made this possible for registered user where user enters the number of top trending anime and the result based on popularity is fetched. The results (top 3 trending anime) is shown in the fig.27

Registered Users are able to modify the ratings of anime which they have previously rated. Here Registered User with ID 5 has rated the anime as shown in Fig 28. Now, the user wants to modify the anime named "Shuffle!" the user is asked

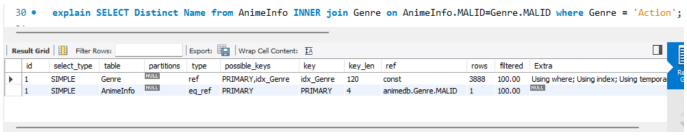

Fig. 26. 'EXPLAIN' command on search query based on Genre

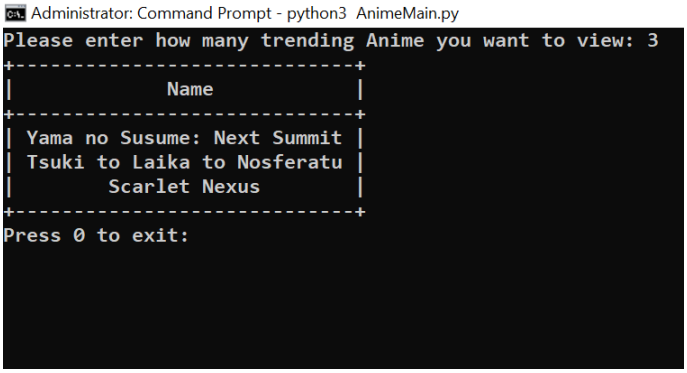

Fig. 27. search results of top 3 trending anime

to enter the new value as shown in fig. 29 and the updated result is shown in fig 30

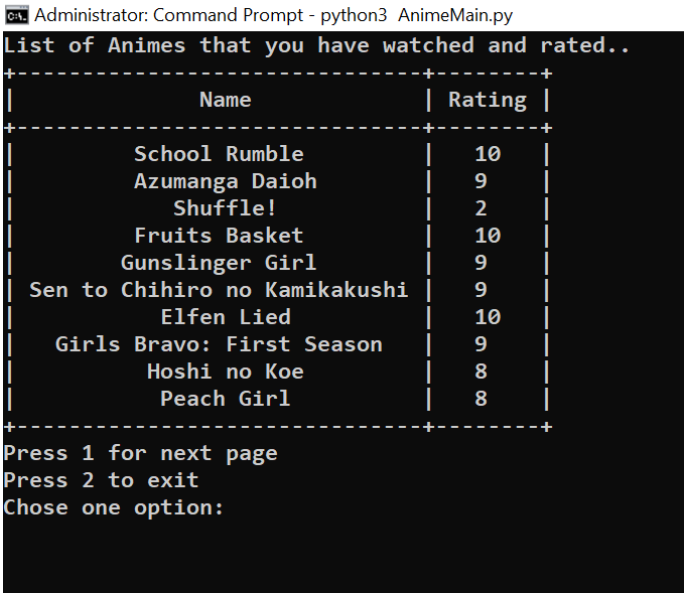

Fig. 28. List of Anime rated by user 5

C. Testing for Admin Users

In our project we have two DB admin with UserID 0 and UserID 1. Their role is 'Admin User'. On logging in with admin user id and password. Admin user is able to insert a new anime record, update current anime record. The DB admin can delete a particular record. In our case we are deleting a record from User database.

Fig.31 shows the list of available choices the admin wishes to alter the record in the database. We have added a check

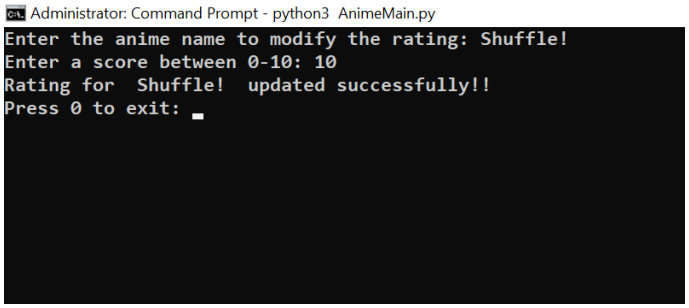

Fig. 29. Anime rating to be modified with new rating

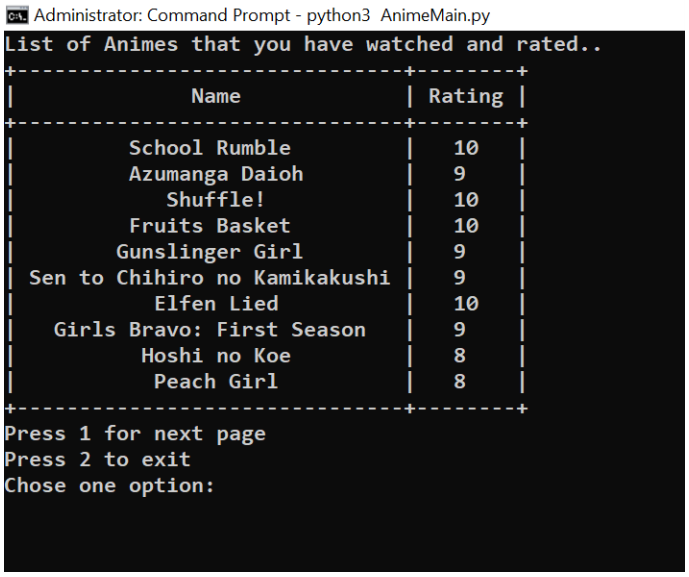

Fig. 30. Modified rating from 2 to 10 for anime 'Shuffle!'

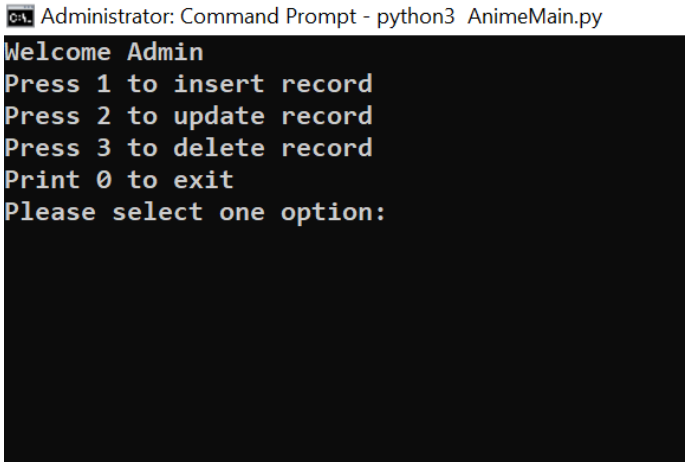

Fig. 31. Choices for Admin User'

constraint named 'CHK_MALID' which checks that MALID should always be greater than zero. The DB admin is trying to enter the value of MALID say '0' as shown in fig. 32. The check constraint is violated and so an error from MySQL is fetched and printed on the CMD stating the admin to enter correct value.

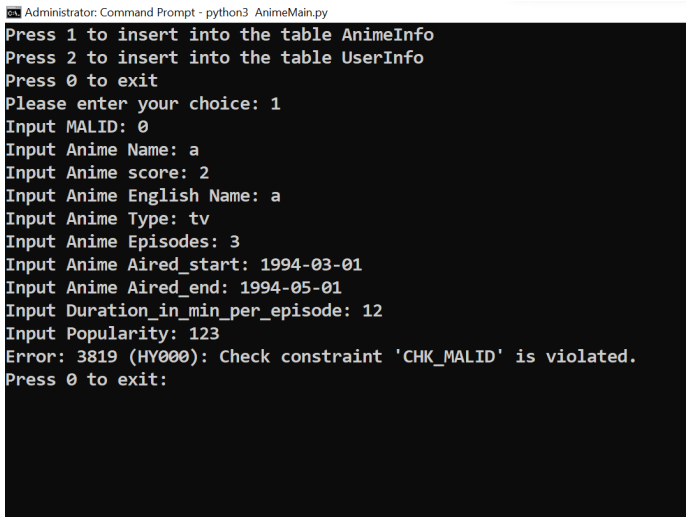

Fig. 32. Inserting incorrect anime record

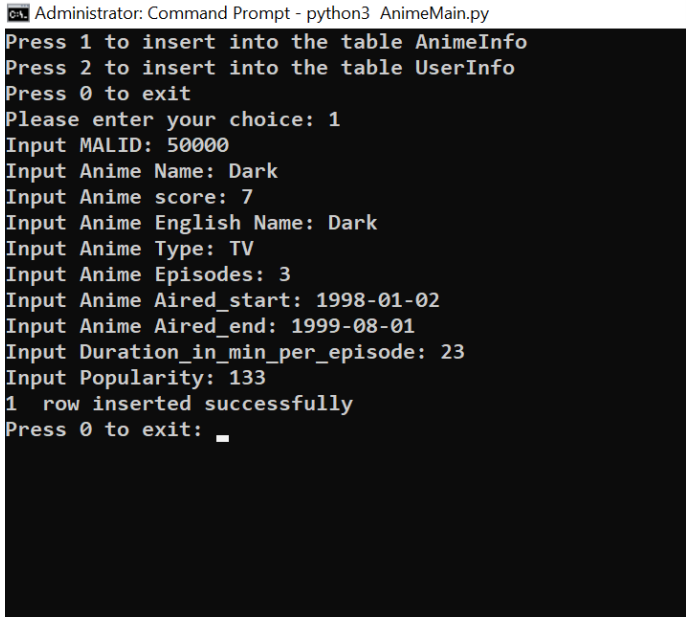

Fig. 33. Inserting a new anime record

Fig.33 shows insertion of a new anime record in the database. Fig 34 shows updating the anime from 'Dark' to 'Dark Knight'

Fig. 35 shows deleting the user from user database.

IV. Future Scope

In this section we will mention some future directions that could be worked on. The following features will be added to our existing project:

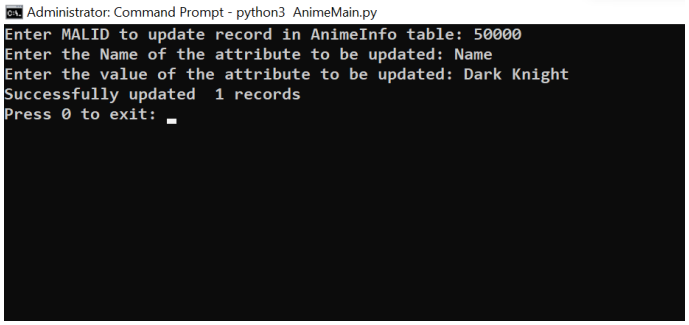

Fig. 34. Modifying Anime record

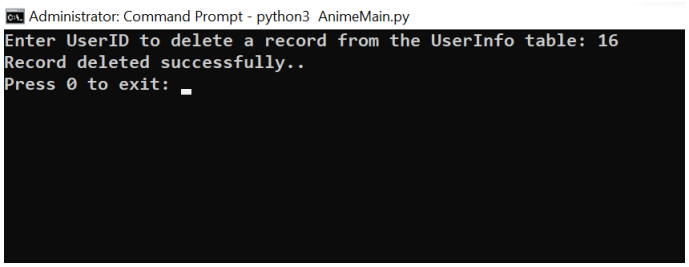

Fig. 35. Deleting user data from user database

1) Data entry operator: These users will add information about a new anime into the DB system and will only have add privilege to the DB tables.

2) Auditor: These users will do auditing on the anime recommendation system. These users will decide whether to keep an anime in the system or to remove it from the system based on some parameters (reviews, popularity). The users will have remove access to some DB tables.

3) Developer: These users will be able to modify code in the system and will have add/ remove/ modify/alter access to the DB.

4) Data-Mining: Data mining implementation in our project will help our recommendation system to find hidden relationships (for example between watching status and Scores etc) and help to any client user to get better recommendations.

V. Conclusion

In this work we have successfully designed a database that can be used to search the anime data based on various search criteria like name, popularity, genre etc. This will help users to get better recommendation. The database created can be altered using DB admin to keep data updated.

References

[1] Valdivieso, H. (2021, July 13). Anime recommendation database 2020. Kaggle. Retrieved December 23, 2021, from https://www.kaggle.com/hernan4444/anime-recommendation-database-2020?select=watching_status.csv