# zllor1uqs

February 15, 2023

```
[3]: from google.colab import drive
     drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[1]: import pandas as pd
     import sqlite3

     from IPython.display import display, HTML
```

```
[ ]: # Note that this is not the same db we have used in course videos, please␣
      ↪download from this link
     # https://drive.google.com/file/d/10-1-L1DdNxEK6O6nG2jS31MbrMh-OnXM/view?
      ↪usp=sharing
```

```
[2]: conn = sqlite3.connect("/content/drive/MyDrive/Colab Notebooks/
      ↪Db-IMDB-Assignment.db")
```

## Overview of all tables

```
[ ]: tables = pd.read_sql_query("SELECT NAME AS 'Table_Name' FROM sqlite_master␣
      ↪WHERE type='table'",conn)
     tables = tables["Table_Name"].values.tolist()
```

```
[ ]: for table in tables:
         query = "PRAGMA TABLE_INFO({})".format(table)
         schema = pd.read_sql_query(query,conn)
         print("Schema of",table)
         display(schema)
         print("-"*100)
         print("\n")
```

```
Schema of Movie

   cid     name     type  notnull dflt_value  pk
0    0    index  INTEGER        0       None   0
1    1      MID     TEXT        0       None   0
2    2    title     TEXT        0       None   0
3    3     year     TEXT        0       None   0
```

```
4    4     rating     REAL        0       None   0
5    5  num_votes  INTEGER        0       None   0
--------------------------------------------------------------------------------
--------------------


Schema of Genre

   cid   name     type  notnull dflt_value  pk
0    0  index  INTEGER        0       None   0
1    1   Name     TEXT        0       None   0
2    2    GID  INTEGER        0       None   0
--------------------------------------------------------------------------------
--------------------


Schema of Language

   cid   name     type  notnull dflt_value  pk
0    0  index  INTEGER        0       None   0
1    1   Name     TEXT        0       None   0
2    2   LAID  INTEGER        0       None   0
--------------------------------------------------------------------------------
--------------------


Schema of Country

   cid   name     type  notnull dflt_value  pk
0    0  index  INTEGER        0       None   0
1    1   Name     TEXT        0       None   0
2    2    CID  INTEGER        0       None   0
--------------------------------------------------------------------------------
--------------------


Schema of Location

   cid   name     type  notnull dflt_value  pk
0    0  index  INTEGER        0       None   0
1    1   Name     TEXT        0       None   0
2    2    LID  INTEGER        0       None   0
--------------------------------------------------------------------------------
--------------------


Schema of M_Location
```

```
   cid   name     type  notnull dflt_value  pk
0    0  index  INTEGER        0       None   0
1    1    MID     TEXT        0       None   0
2    2    LID     REAL        0       None   0
3    3     ID  INTEGER        0       None   0
--------------------------------------------------------------------------------
--------------------
```

```
Schema of M_Country
   cid   name     type  notnull dflt_value  pk
0    0  index  INTEGER        0       None   0
1    1    MID     TEXT        0       None   0
2    2    CID     REAL        0       None   0
3    3     ID  INTEGER        0       None   0
--------------------------------------------------------------------------------
--------------------
```

```
Schema of M_Language
   cid   name     type  notnull dflt_value  pk
0    0  index  INTEGER        0       None   0
1    1    MID     TEXT        0       None   0
2    2   LAID  INTEGER        0       None   0
3    3     ID  INTEGER        0       None   0
--------------------------------------------------------------------------------
--------------------
```

```
Schema of M_Genre
   cid   name     type  notnull dflt_value  pk
0    0  index  INTEGER        0       None   0
1    1    MID     TEXT        0       None   0
2    2    GID  INTEGER        0       None   0
3    3     ID  INTEGER        0       None   0
--------------------------------------------------------------------------------
--------------------
```

```
Schema of Person
   cid    name     type  notnull dflt_value  pk
0    0   index  INTEGER        0       None   0
1    1     PID     TEXT        0       None   0
```

```
2    2    Name     TEXT        0       None   0
3    3  Gender     TEXT        0       None   0
--------------------------------------------------------------------------------
--------------------
```

```
Schema of M_Producer
   cid   name     type  notnull dflt_value  pk
0    0  index  INTEGER        0       None   0
1    1    MID     TEXT        0       None   0
2    2    PID     TEXT        0       None   0
3    3     ID  INTEGER        0       None   0
--------------------------------------------------------------------------------
--------------------
```

```
Schema of M_Director
   cid   name     type  notnull dflt_value  pk
0    0  index  INTEGER        0       None   0
1    1    MID     TEXT        0       None   0
2    2    PID     TEXT        0       None   0
3    3     ID  INTEGER        0       None   0
--------------------------------------------------------------------------------
--------------------
```

```
Schema of M_Cast
   cid   name     type  notnull dflt_value  pk
0    0  index  INTEGER        0       None   0
1    1    MID     TEXT        0       None   0
2    2    PID     TEXT        0       None   0
3    3     ID  INTEGER        0       None   0
--------------------------------------------------------------------------------
--------------------
```

## 0.1 Q1 — List all the directors who directed a 'Comedy' movie in a leap year. (You need to check that the genre is 'Comedy' and year is a leap year) Your query should return director name, the movie name, and the year.

To determine whether a year is a leap year, follow these steps:

STEP-1: If the year is evenly divisible by 4, go to step 2. Otherwise, go to step 5.

STEP-2: If the year is evenly divisible by 100, go to step 3. Otherwise, go to step 4.

STEP-3: If the year is evenly divisible by 400, go to step 4. Otherwise, go to step 5.

STEP-4: The year is a leap year (it has 366 days).

STEP-5: The year is not a leap year (it has 365 days).

Year 1900 is divisible by 4 and 100 but it is not divisible by 400, so it is not a leap year.

[ ]:
```
%%time
def grader_1(q1):
    q1_results  = pd.read_sql_query(q1,conn)
    print(q1_results.head(10))
    assert (q1_results.shape == (232,3))

query1 = """
Select p.Name Director_Name, m.title Movie_Name , m.Year Year from Person p ␣
 ↪join M_director md On TRIM(md.Pid) = TRIM(p.PID)
join Movie m on md.MID = m.MID  join (select MID, GID from M_genre Where Gid In␣
 ↪(Select Gid from Genre where
Name Like "%Comedy%" )) G On G.MID=m.MID where (CAST(SUBSTR(TRIM(year),-4) AS␣
 ↪INTEGER) %4=0)

 """
grader_1(query1)
```

```
       Director_Name                          Movie_Name  Year
0        Milap Zaveri                          Mastizaade  2016
1        Danny Leiner  Harold & Kumar Go to White Castle  2004
2      Anurag Kashyap                  Gangs of Wasseypur  2012
3        Frank Coraci         Around the World in 80 Days  2004
4       Griffin Dunne              The Accidental Husband  2008
5         Anurag Basu                              Barfi!  2012
6     Gurinder Chadha                  Bride & Prejudice  2004
7          Mike Judge     Beavis and Butt-Head Do America  1996
8   Tarun Mansukhani                             Dostana  2008
9        Shakun Batra                       Kapoor & Sons  2016
CPU times: user 24.5 s, sys: 16.1 ms, total: 24.5 s
Wall time: 24.9 s
```

## 0.2 Q2 — List the names of all the actors who played in the movie 'Anand' (1971)

[ ]:
```
%%time
def grader_2(q2):
    q2_results  = pd.read_sql_query(q2,conn)
    print(q2_results.head(10))
    assert (q2_results.shape == (17,1))
```

```
query2 = """ select p.Name from Person p join M_cast m On Trim(p.PID) = Trim(m.
 ↪PID) join
 Movie V On V.MID=m.MID where title = 'Anand' AND year='1971' """
grader_2(query2)
```

```
                 Name
0   Amitabh Bachchan
1      Rajesh Khanna
2      Sumita Sanyal
3         Ramesh Deo
4          Seema Deo
5     Asit Kumar Sen
6         Dev Kishan
7       Atam Prakash
8      Lalita Kumari
9             Savita
CPU times: user 439 ms, sys: 19 ms, total: 459 ms
Wall time: 470 ms
```

## 0.3 Q3 — List all the actors who acted in a film before 1970 and in a film after 1990. (That is: < 1970 and > 1990.)

[ ]:
```
%%time
def grader_3(q3):
    q3_results  = pd.read_sql_query(q3,conn)
    print(q3_results.head(10))
    assert (q3_results.shape == (300,1))

query3 = """
SELECT DISTINCT P.Name FROM Person P
WHERE TRIM(P.PID) IN
(SELECT DISTINCT TRIM(mc.PID) FROM M_Cast mc
WHERE TRIM(mc.MID) IN
(SELECT DISTINCT TRIM(m.MID) FROM Movie m
WHERE CAST(SUBSTR(m.year,-4) AS INTEGER)<'1970'))
AND TRIM(p.PID) IN
(SELECT DISTINCT TRIM(mc2.PID) FROM M_Cast mc2
 WHERE TRIM(mc2.MID) IN
(SELECT DISTINCT TRIM(m2.MID) FROM Movie m2
WHERE CAST(SUBSTR(m2.year,-4) AS INTEGER)>'1990'))  """
grader_3(query3)
```

```
                 Name
0       Rishi Kapoor
1   Amitabh Bachchan
```

```
2             Asrani
3       Zohra Sehgal
4    Parikshat Sahni
5      Rakesh Sharma
6        Sanjay Dutt
7          Ric Young
8             Yusuf
9     Suhasini Mulay
CPU times: user 154 ms, sys: 4 ms, total: 158 ms
Wall time: 165 ms
```

## 0.4 Q4 — List all directors who directed 10 movies or more, in descending order of the number of movies they directed. Return the directors' names and the number of movies each of them directed.

[ ]:
```
%%time

def grader_4a(query_4a):
    query_4a = pd.read_sql_query(query_4a,conn)
    print(query_4a.head(10))
    return (query_4a.shape == (1462,2))

query_4a =""" Select Name Director_Name, count(title) No_of_Movie from Person p␣
 ↪join M_director m on Trim(p.PID) = Trim(m.PID)
 join Movie V on m.MID=V.MID Group By p.name having Count(title)>10 order by␣
 ↪No_of_Movie Desc"""
print(grader_4a(query_4a))

# using the above query, you can write the answer to the given question
```

```
           Director_Name  No_of_Movie
0          David Dhawan           39
1          Mahesh Bhatt           36
2          Priyadarshan           30
3        Ram Gopal Varma           30
4          Vikram Bhatt           29
5    Hrishikesh Mukherjee         27
6            Yash Chopra           21
7         Shakti Samanta           19
8        Basu Chatterjee           19
9          Subhash Ghai           18
False
CPU times: user 26.2 s, sys: 15 ms, total: 26.3 s
Wall time: 26.4 s
```

[ ]:
```
%%time
def grader_4(q4):
    q4_results  = pd.read_sql_query(q4,conn)
    print(q4_results.head(10))
    assert (q4_results.shape == (58,2))

query4 = """ Select Name Director_Name, count(*) No_of_Movie from Person p join␣
 ↪M_director m on Trim(p.PID) = Trim(m.PID)
 join Movie V on m.MID=V.MID Group By p.name having No_of_Movie>=10 order by␣
 ↪No_of_Movie Desc """
grader_4(query4)
```

```
          Director_Name  No_of_Movie
0          David Dhawan           39
1          Mahesh Bhatt           36
2          Priyadarshan           30
3        Ram Gopal Varma          30
4           Vikram Bhatt          29
5   Hrishikesh Mukherjee          27
6            Yash Chopra          21
7         Shakti Samanta          19
8        Basu Chatterjee          19
9           Subhash Ghai          18
CPU times: user 26.4 s, sys: 20.3 ms, total: 26.4 s
Wall time: 26.5 s
```

## 0.5 Q5.a — For each year, count the number of movies in that year that had only female actors.

### 0.5.1 LOGIC :

Select Year and its count from Movie Table , Gett Pid NO. of actor from M_cast and connect it to person table to get Pid NO. of only Female Actor And Exclude that movie in which male actor had worked By Using "Not IN" And In last Group It by Year

```
[ ]: %%time
def grader_5a(q5a):
    q5a_results  = pd.read_sql_query(q5a,conn)
    print(q5a_results.head(10))
    assert (q5a_results.shape == (4,2))

query5a = """Select Year, Count(MID) No_Of_Movie From Movie Where MId In␣
 ↪(Select Mid From M_cast Where Trim(Pid) In
(Select Pid From Person Where Gender = "Female"))
And Mid Not IN (Select Mid From M_cast where Trim(Pid) In (Select Pid From␣
 ↪Person Where Gender = "Male")) Group By Year """
grader_5a(query5a)
```

```
   year  No_Of_Movie
0  1939            1
```

```
1    1999           1
2    2000           1
3  I 2018           1
CPU times: user 171 ms, sys: 6.29 ms, total: 177 ms
Wall time: 179 ms
```

## 0.6 Q5.b — Now include a small change: report for each year the percentage of movies in that year with only female actors, and the total number of movies made that year. For example, one answer will be: 1990 31.81 13522 meaning that in 1990 there were 13,522 movies, and 31.81% had only female actors. You do not need to round your answer.

**Logic :** We have created Two Table Named T1 , T2 and join both of them. T1 Stores year and no. of movies in that year in which only female actor had acted. Similarly, T2 stores each distinct year and no. of movies in that year. From Both tables We Select Year from T1, percentage (No. of movie in which female acted * 100/ Total no. of movies) and Total movies. And Hence we get the result as require.

[14]:
```
%%time
def grader_5b(q5b):
    q5b_results  = pd.read_sql_query(q5b,conn)
    print(q5b_results.head(10))
    assert (q5b_results.shape == (4,3))
query5b = """
Select T1.Year Year, ((T1.Movie_Count*100.0)/T2.Total_Movie) Percent , T2.
 ↪Total_Movie Total_Movie
From (Select CAST(SUBSTR(m.year,-4) AS INTEGER) Year, Count(*) Movie_Count From␣
 ↪Movie m where Mid Not In (Select mc.Mid
From M_cast mc join Person p On p.Pid=Trim(mc.Pid) Group By Trim(mc.Mid), p.
 ↪Gender Having p.Gender = "Male" ) Group BY year) T1 Join
(Select m.Year, Count(*) Total_Movie From Movie m Group By m.year) T2
On T1.Year = T2.Year """
grader_5b(query5b)
```

```
   Year    Percent  Total_Movie
0  1939  50.000000            2
1  1999   1.515152           66
2  2000   1.562500           64
3  2018   1.075269           93
CPU times: user 136 ms, sys: 4.67 ms, total: 141 ms
Wall time: 144 ms
```

## 0.7 Q6 — Find the film(s) with the largest cast. Return the movie title and the size of the cast. By “cast size” we mean the number of distinct actors that played in that movie: if an actor played multiple roles, or if it simply occurs multiple times in casts, we still count her/him only once.

[ ]:
```
%%time
def grader_6(q6):
    q6_results  = pd.read_sql_query(q6,conn)
    print(q6_results.head(10))
    assert (q6_results.shape == (3473, 2))

query6 = """
Select m.Title , Count(Trim(C.Pid)) Cast_Size From Movie m join M_Cast C
on m.MID = C.MID  GROUP BY m.MID order by Cast_Size DESC
"""
grader_6(query6)
```

```
                        title  Cast_Size
0                Ocean's Eight        238
1                     Apaharan        233
2                         Gold        215
3               My Name Is Khan        213
4  Captain America: Civil War        191
5                     Geostorm        170
6                      Striker        165
7                         2012        154
8                       Pixels        144
9        Yamla Pagla Deewana 2        140
CPU times: user 171 ms, sys: 16.1 ms, total: 187 ms
Wall time: 193 ms
```

### 0.7.1 Q7 — A decade is a sequence of 10 consecutive years.

### 0.7.2 For example, say in your database you have movie information starting from 1931.

### 0.7.3 the first decade is 1931, 1932, ..., 1940,

### 0.7.4 the second decade is 1932, 1933, ..., 1941 and so on.

### 0.7.5 Find the decade D with the largest number of films and the total number of films in D

[ ]:
```
%%time
def grader_7(q7):
    q7_results  = pd.read_sql_query(q7,conn)
    print(q7_results.head(10))
    assert (q7_results.shape == (1, 2))
```

```
query7 = """SELECT d.year Start_Of_Decade, count(title) No_of_Movies FROM␣
 ↪(SELECT DISTINCT year from Movie) d JOIN Movie m
 ON CAST(SUBSTR(m.year,-4) AS INTEGER) >= d.year and CAST(SUBSTR(m.year,-4) AS␣
 ↪INTEGER)<= d.year+9 GROUP BY d.year+9
  ORDER BY No_of_Movies desc LIMIT 1"""
grader_7(query7)
# if you check the output we are printinng all the year in that decade, its␣
 ↪fine you can print 2008 or 2008-2017
```

```
  Start_Of_Decade  No_of_Movies
0            2008          1203
CPU times: user 121 ms, sys: 993 µs, total: 122 ms
Wall time: 128 ms
```

## 0.8 Q8 — Find all the actors that made more movies with Yash Chopra than any other director.

[ ]:
```
%%time
def grader_8a(q8a):
    q8a_results  = pd.read_sql_query(q8a,conn)
    print(q8a_results.head(10))
    assert (q8a_results.shape == (73408, 3))

query8a = """
Select mc.PID Actor_ID , md.PID Director_ID, COUNT(mc.mid) Movie_Count From␣
 ↪M_Cast mc join M_Director
md ON mc.mid = md.mid Group by mc.pid, md.pid
"""
grader_8a(query8a)

# using the above query, you can write the answer to the given question
```

```
     Actor_ID Director_ID  Movie_Count
0   nm0000002   nm0496746            1
1   nm0000027   nm0000180            1
2   nm0000039   nm0896533            1
3   nm0000042   nm0896533            1
4   nm0000047   nm0004292            1
5   nm0000073   nm0485943            1
6   nm0000076   nm0000229            1
7   nm0000092   nm0178997            1
8   nm0000093   nm0000269            1
9   nm0000096   nm0113819            1
CPU times: user 256 ms, sys: 20.9 ms, total: 277 ms
Wall time: 279 ms
```

```
[ ]: %%time

def grader_8(q8):
    q8_results  = pd.read_sql_query(q8,conn)
    print(q8_results.head(10))
    print(q8_results.shape)
    assert (q8_results.shape == (245, 2))

query8 = """
With T1 As (Select Trim(p.pid) ID From Person p Where Trim(Name) = "Yash␣
 ↪Chopra"),

T2 As (Select Trim(mc.PID) Actor_ID , Trim(md.PID) Director_ID, COUNT(distinct␣
 ↪mc.mid) Movie_Count From M_Cast mc join M_Director md
 ON mc.mid = md.mid Group by Trim(mc.PID) , Trim(md.PID)),

 T3 As (Select T2.Actor_ID, T2.Director_ID, Movie_Count From T1, T2 Where T1.ID␣
 ↪= T2.Director_ID),

 T4 As (Select T2.Actor_ID, T2.Director_ID, Movie_Count From T1, T2 Where T1.ID␣
 ↪!= T2.Director_ID)

 Select p.Name , T3.Movie_Count  From Person p join T3 On Trim(p.pid) = T3.
 ↪Actor_ID
 where not exists (Select 1 from T4 where T3.Actor_ID = T4.Actor_ID and T3.
 ↪Movie_Count < T4.Movie_Count)
"""
grader_8(query8)
```

```
                 Name  Movie_Count
0       Sharib Hashmi            1
1     Kulbir Badesron            1
2         Gurdas Maan            1
3     Parikshat Sahni            3
4       Claire Ashton            1
5      Waheeda Rehman            5
6            Taj Gill            1
7          Kumud Pant            1
8    Gerald Tomkinson            1
9    Dev K. Kantawall            1
(245, 2)
CPU times: user 1.73 s, sys: 24.7 ms, total: 1.75 s
Wall time: 1.75 s
```

## 0.9 Q9 — The Shahrukh number of an actor is the length of the shortest path between the actor and Shahrukh Khan in the "co-acting" graph. That is, Shahrukh Khan has Shahrukh number 0; all actors who acted in the same film as Shahrukh have Shahrukh number 1; all actors who acted in the same film as some actor with Shahrukh number 1 have Shahrukh number 2, etc. Return all actors whose Shahrukh number is 2.

**Logic :** Create Srk 1 Table In which we have Pid no. of those actor who acted with shahrukh khan (exclude shahrukh khan pid no.)

again create srk 2 Table that have pid no. of actor who acted with actor of srk 1 table and after this exclude pid no. of actor in srk1 and srk itself.

[3]:
```
%%time
def grader_9(q9):
    q9_results  = pd.read_sql_query(q9,conn)
    print(q9_results.head(10))
    print(q9_results.shape)
    assert (q9_results.shape == (25698, 1))

query9 = """ with Srk_1 As (Select distinct Trim(mt.pid) pid From (Select p.
 ↪pid, ma.mid S_Id From Person P join M_cast ma on p.pid= Trim(ma.Pid)
Where Trim(p.name) like "%Shah Rukh%" ) T1 Join M_cast mt on T1.S_Id = mt.mid␣
 ↪where T1.pid != Trim(mt.pid)),

Srk_2 As (Select ss.pid pid, md.mid Srk2_mid From M_cast md Join Srk_1  ss On␣
 ↪Trim(md.Pid)=Trim(ss.pid))

Select p.Name Actor_Name From Person P Where p.pid In (Select Trim(mc.Pid) From␣
 ↪M_cast mc Where mc.MId In (Select Srk2_mid From Srk_2))
And p.pid Not In (Select pid From Srk_1) And p.Pid NOt IN (Select pid From␣
 ↪Person  Where Name Like "%Shah Rukh%")"""
grader_9(query9)
```

```
               Actor_Name
0           Freida Pinto
1            Rohan Chand
2           Damian Young
3        Waris Ahluwalia
4    Caroline Christl Long
5          Rajeev Pahuja
6       Michelle Santiago
7         Alicia Vikander
8           Dominic West
9          Walton Goggins
(25698, 1)
CPU times: user 39.1 s, sys: 75.5 ms, total: 39.2 s
Wall time: 39.4 s
```