SQL Assignment

In [1]:
```
from google.colab import files

files = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Db-IMDB-Assignment.db to Db-IMDB-Assignment.db
```

In [2]:
```
import pandas as pd
import sqlite3

from IPython.display import display, HTML
```

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

In [3]:
```
conn = sqlite3.connect("/content/Db-IMDB-Assignment.db")
```

Overview of all tables

In [4]:
```
tables = pd.read_sql_query("SELECT NAME AS 'Table_Name' FROM sqlite_master WHERE type='t
tables = tables["Table_Name"].values.tolist()
```

In [ ]:
```
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

Useful tips:

1. the year column in 'Movie' table, will have few chracters other than numbers which you need to be preprocessed, you need to get a substring of last 4 characters, its better if you convert it as int type, ex: CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER)
2. For almost all the TEXT columns we have show, please try to remove trailing spaces, you need to use TRIM() function
3. When you are doing count(coulmn) it won't consider the "NULL" values, you might need to explore other alternatives like Count(*)

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.

In [ ]:
```
%%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
                DISTINCT p.name AS Director,
                m.title AS Movie_Name,
                CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER) AS yr
             FROM Movie AS m
             INNER JOIN M_Director AS md ON m.MID = md.MID
             INNER JOIN Person AS p ON md.PID = p.PID
             INNER JOIN M_Genre AS mg ON m.MID = mg.MID
             INNER JOIN Genre AS g ON g.GID = mg.GID
             WHERE (g.name LIKE '%Comedy%')
             AND (yr%4 = 0)"""
grader_1(query1)
```

```
             Director                          Movie_Name    yr
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 65.1 ms, sys: 1.28 ms, total: 66.3 ms
Wall time: 66 ms
```

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

In [ ]:
```
%%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 AS Actors
            FROM Person AS P
            WHERE p.PID IN (SELECT TRIM(mc.PID) FROM M_Cast AS mc LEFT JOIN Movie as m O
                  WHERE m.title = 'Anand') """
grader_2(query2)
```

```
              Actors
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 62.6 ms, sys: 2.08 ms, total: 64.6 ms
Wall time: 66.5 ms
```

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

In [ ]:
```
%%time

def grader_3a(query_less_1970, query_more_1990):
    q3_a = pd.read_sql_query(query_less_1970,conn)
    print(q3_a.shape)
    q3_b = pd.read_sql_query(query_more_1990,conn)
    print(q3_b.shape)
    return (q3_a.shape == (4942,1)) and (q3_b.shape == (62570,1))

query_less_1970 ="""
Select p.PID from Person p
inner join
(
    select trim(mc.PID) PD, mc.MID from M_cast mc
where mc.MID
in
(
    select mv.MID from Movie mv where CAST(SUBSTR(mv.year,-4) AS Integer)<1970
)
) r1
on r1.PD=p.PID
"""
query_more_1990 ="""
Select p.PID from Person p
inner join
(
    select trim(mc.PID) PD, mc.MID from M_cast mc
where mc.MID
in
(
    select mv.MID from Movie mv where CAST(SUBSTR(mv.year,-4) AS Integer)>1990
)
) r1
on r1.PD=p.PID """
print(grader_3a(query_less_1970, query_more_1990))

# using the above two queries, you can find the answer to the given question
```

```
(4942, 1)
(62570, 1)
True
CPU times: user 276 ms, sys: 11.8 ms, total: 288 ms
Wall time: 290 ms
```

SELECT p.name AS Actor_Name FROM Person AS p JOIN M_Cast AS mc ON TRIM(mc.PID) = p.PID JOIN Movie AS m ON m.MID = mc.MID WHERE CAST(SUBSTR(m.year,-4) AS Integer) < 1970 INTERSECT SELECT p.name AS Actor_Name FROM Person AS p JOIN M_Cast AS mc ON TRIM(mc.PID) = p.PID JOIN Movie AS m ON m.MID = mc.MID WHERE CAST(SUBSTR(m.year,-4) AS Integer) > 1990

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

```
    assert (q3_results.shape == (300,1))

query3 = """WITH

            BEFORE_1970 AS
            (SELECT DISTINCT TRIM(MC.PID) AS PID
            FROM Movie AS m
            INNER JOIN M_Cast AS mc
            ON m.MID = mc.MID
            WHERE CAST(SUBSTR(m.year,-4) AS Integer) < 1970),

            AFTER_1990 AS
            (SELECT DISTINCT TRIM(MC.PID) AS PID
            FROM Movie AS m
            INNER JOIN M_Cast AS mc
            ON m.MID = mc.MID
            WHERE CAST(SUBSTR(m.year,-4) AS Integer) > 1990)

            SELECT DISTINCT TRIM(p.Name) AS Actor_Name
            FROM BEFORE_1970 AS b
            INNER JOIN AFTER_1990 AS a
            ON a.PID = b.PID
            INNER JOIN Person AS p
            ON  a.PID = TRIM(P.PID)
             """
grader_3(query3)
```

```
         Actor_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
(300, 1)
CPU times: user 342 ms, sys: 9.68 ms, total: 352 ms
Wall time: 354 ms
```

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.

In [ ]:

```
%%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 =""" WITH
              temp AS
              (SELECT TRIM(md.PID) AS PID, COUNT(m.title) AS movies
              FROM Movie AS m
              INNER JOIN M_Director AS md
              ON m.MID = md.MID
              GROUP BY md.PID)

              SELECT p.name AS Director, d.movies
```

```
                FROM Person AS p
                INNER JOIN temp AS d
                ON TRIM(p.PID) = d.PID
                ORDER BY movies DESC"""

print(grader_4a(query_4a))

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

```
                Director  movies
0           David Dhawan      39
1           Mahesh Bhatt      35
2        Ram Gopal Varma      30
3           Priyadarshan      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
True
CPU times: user 39.3 ms, sys: 0 ns, total: 39.3 ms
Wall time: 43.1 ms
```

In [ ]:
```
%%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 = """ WITH
                temp AS
                (SELECT TRIM(md.PID) AS PID, COUNT(m.title) AS movies
                FROM Movie AS m
                INNER JOIN M_Director AS md
                ON m.MID = md.MID
                GROUP BY md.PID)

                SELECT p.name AS Director, d.movies
                FROM Person AS p
                INNER JOIN temp AS d
                ON TRIM(p.PID) = d.PID
                ORDER BY movies DESC
                LIMIT 58 """
grader_4(query4)
```

```
                Director  movies
0           David Dhawan      39
1           Mahesh Bhatt      35
2        Ram Gopal Varma      30
3           Priyadarshan      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 38.3 ms, sys: 0 ns, total: 38.3 ms
Wall time: 42.1 ms
```

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

In [ ]:
```
%%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 CAST(SUBSTR(m.year,-4) AS Integer) AS yr, COUNT(m.MID) AS Female_Ca
              FROM M_CAST AS mc
              INNER JOIN Movie AS m
              ON mc.MID = m.MID
              WHERE m.MID NOT IN  (SELECT DISTINCT mc.MID AS mv
                                   FROM M_Cast AS mc
                                   INNER JOIN Person AS p
                                   ON p.PID = TRIM(mc.PID)
                                   WHERE p.Gender IN ('Male', 'None')
                                   GROUP BY mc.MID, p.Gender)
              GROUP BY yr
               """
grader_5a(query5a)
```

```
     yr  Female_Cast_only_movies
0  1939                        1
1  1999                       11
2  2000                       10
3  2018                        2
CPU times: user 251 ms, sys: 2.05 ms, total: 253 ms
Wall time: 255 ms
```

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.

In [ ]:
```
%%time
def grader_5b(q5b):
    q5b_results  = pd.read_sql_query(q5b,conn)
    print(q5b_results.head(10))
    assert (q5b_results.shape == (4,3))
# the multiplication by 1.0 is done to get the query to return float values instead of n
query5b = """ WITH

              total_movies AS
              (SELECT COUNT(DISTINCT(MID)) AS tot_mv, CAST(SUBSTR(year,-4) AS Integer) A
              FROM Movie
              GROUP BY yr),

              females_only AS
              (SELECT CAST(SUBSTR(m.year,-4) AS Integer) AS yr, COUNT(m.MID) AS Female_C
              FROM M_CAST AS mc
              INNER JOIN Movie AS m
              ON mc.MID = m.MID
              WHERE m.MID NOT IN  (SELECT DISTINCT mc.MID AS mv
                                   FROM M_Cast AS mc
                                   INNER JOIN Person AS p
                                   ON p.PID = TRIM(mc.PID)
                                   WHERE p.Gender IN ('Male', 'None')
                                   GROUP BY mc.MID, p.Gender)
              GROUP BY yr)

              SELECT f.yr, (f.Female_Cast_only_movies*1.0)/t.tot_mv AS Percentage_Female
              FROM females_only AS f
```

```
            LEFT JOIN total_movies As t
            ON f.yr = t.yr
             """
grader_5b(query5b)
```

```
     yr  Percentage_Female_Only_Movie  total_movies
0  1939                      0.500000             2
1  1999                      0.166667            66
2  2000                      0.156250            64
3  2018                      0.019231           104
CPU times: user 255 ms, sys: 1.03 ms, total: 256 ms
Wall time: 257 ms
```

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.

In [ ]:
```
%%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(DISTINCT(mc.PID)) AS count
             FROM Movie AS m
             INNER JOIN M_Cast as mc
             ON mc.MID = m.MID
             GROUP BY m.MID
             ORDER BY count DESC """
grader_6(query6)
```

```
                        title  count
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 194 ms, sys: 16.7 ms, total: 211 ms
Wall time: 211 ms
```

Q7 --- A decade is a sequence of 10 consecutive years.

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

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

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

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

In [ ]:
```
%%time
def grader_7a(q7a):
    q7a_results  = pd.read_sql_query(q7a,conn)
    print(q7a_results.head(10))
    assert (q7a_results.shape == (78, 2))

query7a = """ WITH
              years_and_decades AS
              (SELECT CAST(SUBSTR(m.year,-4) AS Integer) AS yr,
                      CAST(SUBSTR(m.year,-4) AS Integer) AS decade_start,
                      CAST(SUBSTR(m.year,-4) AS Integer)+9 AS decade_stop
               FROM Movie AS m
               ORDER BY yr)

              SELECT CAST(SUBSTR(year,-4) AS Integer) AS Movie_Year,
              COUNT(DISTINCT(MID)) AS Total_Movies
              FROM Movie, years_and_decades
              WHERE  Movie_Year BETWEEN decade_start AND decade_stop
              GROUP BY Movie_Year """
grader_7a(query7a)

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

```
   Movie_Year  Total_Movies
0        1931             1
1        1936             3
2        1939             2
3        1941             1
4        1943             1
5        1946             2
6        1947             2
7        1948             3
8        1949             3
9        1950             2
CPU times: user 3.18 s, sys: 117 ms, total: 3.3 s
Wall time: 3.32 s
```

In [ ]:
```
%%time
def grader_7b(q7b):
    q7b_results  = pd.read_sql_query(q7b,conn)
    print(q7b_results.head(10))
    assert (q7b_results.shape == (713, 4))

query7b = """ WITH
              years_and_decades AS
              (SELECT CAST(SUBSTR(m.year,-4) AS Integer) AS yr,
                      CAST(SUBSTR(m.year,-4) AS Integer) AS decade_start,
                      CAST(SUBSTR(m.year,-4) AS Integer)+9 AS decade_stop
               FROM Movie AS m
               ORDER BY yr),

              decade_movie_count AS
              (SELECT CAST(SUBSTR(year,-4) AS Integer) AS Movie_Year,
              COUNT(DISTINCT(MID)) AS Total_Movies
              FROM Movie, years_and_decades
              WHERE  Movie_Year BETWEEN decade_start AND decade_stop
              GROUP BY Movie_Year)

              SELECT a.Movie_Year, a.Total_Movies, b.Movie_Year, b.Total_Movies
              FROM  decade_movie_count a, decade_movie_count b
              WHERE (a.Movie_Year <= b.Movie_Year)
              AND (b.Movie_Year <= a.Movie_Year+9)

          """
grader_7b(query7b)
```

```
# if you see the below results the first movie year is less than 2nd movie year and
# 2nd movie year is less or equal to the first movie year+9

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

```
   Movie_Year  Total_Movies  Movie_Year  Total_Movies
0        1931             1        1931             1
1        1931             1        1936             3
2        1931             1        1939             2
3        1936             3        1936             3
4        1936             3        1939             2
5        1936             3        1941             1
6        1936             3        1943             1
7        1939             2        1939             2
8        1939             2        1941             1
9        1939             2        1943             1
CPU times: user 6.34 s, sys: 267 ms, total: 6.6 s
Wall time: 6.59 s
```

In [ ]:
```
%%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  COUNT(*) AS Movies_in_the_Decade, m1.year AS Decade
             FROM (SELECT DISTINCT year FROM Movie) m1
             JOIN Movie m2
             ON m1.year <= m2.year AND m2.year <= m1.year+9
             GROUP BY m1.year
             ORDER BY COUNT(*) DESC
             LIMIT 1 """
grader_7(query7)
# if you check the output we are printinng all the year in that decade, its fine you can
```

```
   Movies_in_the_Decade Decade
0                  1126   2008
CPU times: user 79.6 ms, sys: 0 ns, total: 79.6 ms
Wall time: 81.5 ms
```

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

In [ ]:
```
%%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 AS actor, md.PID AS director, COUNT(m.MID)
              FROM Movie AS m
              INNER JOIN M_Director AS md
              ON md.MID = m.MID
              INNER JOIN M_Cast AS mc
              ON mc.MID = m.MID
              GROUP BY actor, director """
grader_8a(query8a)

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

```
       actor   director  COUNT(m.MID)
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 313 ms, sys: 21.8 ms, total: 334 ms
Wall time: 334 ms
```

In [ ]:
```
q8 = """
SELECT PID FROM Person WHERE Name LIKE "%Yash Chopra%"
"""
q8_results  = pd.read_sql_query(q8,conn)
```

In [ ]:
```
q8_results
```

Out[ ]:

	PID
0	nm0007181

In [150…
```
%%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 actor_director AS (SELECT mc.PID AS actor, md.PID AS director, COUNT(*) AS movie_co
FROM M_Director AS md
JOIN M_Cast AS mc
ON TRIM(md.MID) = TRIM(mc.MID)
GROUP BY actor, director),

max_actor_director AS (SELECT DISTINCT actor, director, MAX(movie_counts) AS movies
FROM actor_director
GROUP BY actor),

max_yc_actors AS (SELECT ad.actor, ad.movie_counts
FROM actor_director AS ad, max_actor_director AS mad
WHERE (ad.actor = mad.actor) AND (mad.movies = ad.movie_counts) AND (ad.director LIKE "%

SELECT p.Name AS actor_name, mya.movie_counts AS no_of_movies
FROM max_yc_actors AS mya
JOIN
Person AS p
ON TRIM(mya.actor) = p.PID
ORDER BY no_of_movies DESC


"""
grader_8(query8)
```

```
          actor_name  no_of_movies
0        Jagdish Raj            11
1   Manmohan Krishna            10
2           Iftekhar             9
3      Shashi Kapoor             7
4       Rakhee Gulzar             5
5     Waheeda Rehman              5
6            Ravikant             4
7     Achala Sachdev              4
```

```
8         Neetu Singh              4
9       Leela Chitnis              3
(245, 2)
CPU times: user 48.5 s, sys: 3.42 s, total: 51.9 s
Wall time: 51.7 s
```

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.

In [ ]:
```
q9 = """SELECT PID FROM Person WHERE Name LIKE "%Shah Rukh Khan%" """
pd.read_sql_query(q9,conn)
```

Out[ ]:

	PID
0	nm0451321

In [94]:
```
%%time
def grader_9a(q9a):
    q9a_results  = pd.read_sql_query(q9a,conn)
    print(q9a_results.head(10))
    print(q9a_results.shape)
    assert (q9a_results.shape == (2382, 1))

query9a = """ WITH shahrukhmovies AS (SELECT mc.MID
              FROM Movie AS m
              JOIN M_Cast AS mc
              ON mc.MID = m.MID
              WHERE mc.PID LIKE "%nm0451321%")

              SELECT DISTINCT(mc.PID)
              FROM M_Cast AS mc
              INNER JOIN shahrukhmovies as srm
              ON (srm.MID = mc.MID) AND (mc.PID NOT LIKE "%nm0451321%")



              """
grader_9a(query9a)
# using the above query, you can write the answer to the given question

# selecting actors who acted with srk (S1)
# selecting all movies where S1 actors acted, this forms S2 movies list
# selecting all actors who acted in S2 movies, this gives us S2 actors along with S1 act
# removing S1 actors from the combined list of S1 & S2 actors, so that we get only S2 ac
```

```
          PID
0   nm0004418
1   nm1995953
2   nm2778261
3   nm0631373
4   nm0241935
5   nm0792116
6   nm1300111
```

```
7   nm0196375
8   nm1464837
9   nm2868019
(2382, 1)
CPU times: user 631 ms, sys: 9.02 ms, total: 640 ms
Wall time: 649 ms
```

In [109…

```
%%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 shahrukhmovies AS (SELECT mc.MID
              FROM Movie AS m
              JOIN M_Cast AS mc
              ON mc.MID = m.MID
              WHERE mc.PID LIKE "%nm0451321%"),

            shahrukh1 AS (SELECT DISTINCT(mc.PID)
              FROM M_Cast AS mc
              INNER JOIN shahrukhmovies as srm
              ON (srm.MID = mc.MID) AND (mc.PID NOT LIKE "%nm0451321%")),

            shahrukh1_movies AS (SELECT DISTINCT(mc.MID)
            FROM M_Cast as mc, shahrukh1 AS s1
            WHERE mc.PID = s1.PID),

            shahrukh2 AS (SELECT DISTINCT mc.PID
            FROM M_Cast AS mc
            INNER JOIN shahrukh1_movies AS s1m
            ON mc.MID = s1m.MID
            WHERE (mc.PID NOT IN (SELECT s1.PID FROM shahrukh1 AS s1)) AND (mc.PID NOT L

            SELECT p.Name
            FROM shahrukh2 AS s2
            JOIN
            Person AS p
            WHERE p.PID = TRIM(s2.PID)



              """

grader_9(query9)
```

```
                   Name
0        Alicia Vikander
1           Dominic West
2         Walton Goggins
3              Daniel Wu
4   Kristin Scott Thomas
5           Derek Jacobi
6     Alexandre Willaume
7           Tamer Burjaq
8         Adrian Collins
9         Keenan Arrison
(25698, 1)
CPU times: user 1.53 s, sys: 16 ms, total: 1.55 s
Wall time: 1.57 s
```