SQL Assignment

In [1]:

```
import pandas as pd
import sqlite3

from IPython.display import display, HTML
```

In [2]:

```
# 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
```

In [3]:

```
conn = sqlite3.connect("/content/drive/MyDrive/Colab Notebooks/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
='table'",conn)
tables = tables["Table_Name"].values.tolist()
tables
```

Out[4]:

```
['Movie',
 'Genre',
 'Language',
 'Country',
 'Location',
 'M_Location',
 'M_Country',
 'M_Language',
 'M_Genre',
 'Person',
 'M_Producer',
 'M_Director',
 'M_Cast']
```

In [5]:

```
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 [6]:

```
%%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,M.title,CAST(SUBSTR(TRIM(M.year),-4) AS INTEGER) AS year
               FROM Movie M JOIN M_Director MD ON(MD.mid = M.mid)
               JOIN M_Genre MG ON(MG.mid = M.mid)
               JOIN Genre G ON (MG.gid = G.gid)
               JOIN Person P ON (MD.pid = P.pid)
               WHERE LOWER(G.name) LIKE '%comedy%'
                 AND (
                       (CAST(SUBSTR(TRIM(M.year),-4) AS INTEGER) % 4 = 0
                         AND CAST(SUBSTR(TRIM(M.year),-4) AS INTEGER) % 100 != 0
                       )
                     OR (CAST(SUBSTR(TRIM(M.year),-4) AS INTEGER) % 400 =0
                       )
                 )
               """
grader_1(query1)
```

```
                Name                                 title  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 75 ms, sys: 5.59 ms, total: 80.6 ms
Wall time: 88.8 ms
```

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

In [7]:

```
%%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 MC ON (P.pid = TRIM(MC.pid))
               JOIN  Movie M ON (M.mid = MC.mid)
               WHERE  LOWER(TRIM(M.title)) ='anand'
                 """
grader_2(query2)
```

```
                Name
0   Amitabh Bachchan
1      Rajesh Khanna
2     Brahm Bhardwaj
3         Ramesh Deo
4          Seema Deo
5         Dev Kishan
6        Durga Khote
7      Lalita Kumari
8       Lalita Pawar
9       Atam Prakash
CPU times: user 176 ms, sys: 4.77 ms, total: 181 ms
Wall time: 204 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 [8]:

```
%%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 250 ms, sys: 12.8 ms, total: 263 ms
Wall time: 265 ms
```

In [9]:

```
%%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 = """ SELECT name
               FROM (SELECT  P.pid,P.name
                       FROM  Person P JOIN M_Cast MC ON (P.pid = TRIM(MC.pid))
                       JOIN  Movie M ON (M.mid = MC.mid)
                      WHERE  CAST(SUBSTR(TRIM(M.year),-4) AS INTEGER) > 1990
                  INTERSECT
                     SELECT  P.pid,P.name
                       FROM  Person P JOIN M_Cast MC ON (P.pid = TRIM(MC.pid))
                       JOIN  Movie M ON (M.mid = MC.mid)
                      WHERE  CAST(SUBSTR(TRIM(M.year),-4) AS INTEGER) < 1970
                   )
            """
grader_3(query3)
```

```
                 name
0     Amitabh Bachchan
1   Mohandas K. Gandhi
2                Rekha
3           Dharmendra
4    Prithviraj Kapoor
5        Shammi Kapoor
6        Shashi Kapoor
7        Rajesh Khanna
8          Hema Malini
9          Sanjay Dutt
(300, 1)
CPU times: user 440 ms, sys: 17.6 ms, total: 458 ms
Wall time: 467 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 [10]:

```
%%time

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

query_4a =""" SELECT MD.pid, COUNT(*)
                FROM M_Director MD
                GROUP BY MD.pid
                ORDER BY 2 DESC
               """
print(grader_4a(query_4a))

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

```
         PID  COUNT(*)
0  nm0223522        39
1  nm0080315        35
2  nm0698184        30
3  nm0890060        30
4  nm0080333        29
5  nm0611531        27
6  nm0007181        21
7  nm0154113        19
8  nm0759662        19
9  nm0007131        18
(1462, 2)
True
CPU times: user 11.4 ms, sys: 1.93 ms, total: 13.3 ms
Wall time: 19 ms
```

In [11]:

```
%%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 mov_dir AS (
                SELECT MD.pid, COUNT(*) AS Movie_Count
                  FROM M_Director MD
                GROUP BY MD.pid
                HAVING COUNT(*) >=10
                               )
                SELECT P.name AS Director_Name  , MD.movie_count
                  FROM Person P JOIN mov_dir MD ON (P.PID = MD.PID)
        """
grader_4(query4)
```

```
        Director_Name  Movie_Count
0    Mahesh Manjrekar           15
1    Tigmanshu Dhulia           10
2      Satish Kaushik           12
3      Anurag Kashyap           13
4         Yash Chopra           21
5        Subhash Ghai           18
6       Rakesh Roshan           13
7   Madhur Bhandarkar           12
8         Ketan Mehta           11
9        Mahesh Bhatt           35
CPU times: user 26.3 ms, sys: 0 ns, total: 26.3 ms
Wall time: 27.4 ms
```

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

In [12]:

```
%%time

# note that you don't need TRIM for person table

def grader_5aa(query_5aa):
    query_5aa = pd.read_sql_query(query_5aa,conn)
    print(query_5aa.head(10))
    return (query_5aa.shape == (8846,3))

query_5aa =""" SELECT  MC.mid , P.Gender As Gend, COUNT(*) AS Count
                FROM  M_Cast MC JOIN Person P ON (TRIM(MC.pid) = P.pid)
                GROUP BY MC.mid , P.Gender
 """

print(grader_5aa(query_5aa))

def grader_5ab(query_5ab):
    query_5ab = pd.read_sql_query(query_5ab,conn)
    print(query_5ab.head(10))
    return (query_5ab.shape == (3469, 3))

query_5ab =""" SELECT  MC.mid , P.Gender As Gend, COUNT(*) AS Count
                FROM  M_Cast MC JOIN Person P ON (TRIM(MC.pid) = P.pid)
                WHERE P.Gender <> 'Female'
                GROUP BY MC.mid , P.Gender  """

print(grader_5ab(query_5ab))


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

```
         MID    Gend  Count
0  tt0021594    None      1
1  tt0021594  Female      3
2  tt0021594    Male      5
3  tt0026274    None      2
4  tt0026274  Female     11
5  tt0026274    Male      9
6  tt0027256    None      2
7  tt0027256  Female      5
8  tt0027256    Male      8
9  tt0028217  Female      3
True
         MID  Gend  Count
0  tt0021594  Male      5
1  tt0026274  Male      9
2  tt0027256  Male      8
3  tt0028217  Male      7
4  tt0031580  Male     27
5  tt0033616  Male     46
6  tt0036077  Male     11
7  tt0038491  Male      7
8  tt0039654  Male      6
9  tt0040067  Male     10
True
CPU times: user 356 ms, sys: 9.75 ms, total: 366 ms
Wall time: 369 ms
```

In [13]:

```
%%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(TRIM(M.year),-4)AS INTEGER) as YEAR,
                     COUNT(DISTINCT MC.MID) AS Female_Cast_Only_Movies
                FROM  M_Cast MC JOIN Person P ON (TRIM(MC.pid) = P.pid)
                JOIN  Movie M ON (M.mid = MC.mid)
               WHERE  P.gender = 'Female'
                 AND MC.MID NOT IN (SELECT MC1.MID
                                      FROM M_Cast MC1 JOIN Person P1 ON (TRIM(MC1.pid)
 = P1.pid)
                                       AND  P1.gender <> 'Female'
                                   )
                 GROUP BY CAST(SUBSTR(TRIM(M.year),-4) AS INTEGER)  """
grader_5a(query5a)
```

```
   YEAR  Female_Cast_Only_Movies
0  1939                         1
1  1999                         1
2  2000                         1
3  2018                         1
CPU times: user 188 ms, sys: 2.8 ms, total: 191 ms
Wall time: 191 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 [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 = """ WITH FEMALE_ONLY_MOVIES AS (
                 SELECT CAST(SUBSTR(TRIM(M.year),-4)AS INTEGER) as year,
                        CAST(COUNT(DISTINCT MC.MID) AS FLOAT) AS Female_Cast_Only_Movie
s
                   FROM  M_Cast MC JOIN Person P ON (TRIM(MC.pid) = P.pid)
                   JOIN  Movie M ON (M.mid = MC.mid)
                   WHERE  P.gender = 'Female'
                   AND MC.MID NOT IN (SELECT MC1.MID
                                        FROM M_Cast MC1 JOIN Person P1
                                       ON (TRIM(MC1.pid) = P1.pid)
                                           AND  P1.gender <> 'Female')
                   GROUP BY CAST(SUBSTR(TRIM(M.year),-4)AS INTEGER)
                                       ) ,
               TOTAL_MOVIES AS (
                   SELECT CAST(SUBSTR(TRIM(M.year),-4)AS INTEGER) as year,
                          CAST(COUNT(DISTINCT MC.MID) AS INTEGER) AS Total_Movies
                     FROM  M_Cast MC JOIN Person P ON (TRIM(MC.pid) = P.pid)
                     JOIN  Movie M ON (M.mid = MC.mid)
                     GROUP BY CAST(SUBSTR(TRIM(M.year),-4)AS INTEGER)
                                  )
                    SELECT
                       FOM.year,
                       (FOM.Female_Cast_Only_Movies  / TM.Total_Movies)  AS Percentage_
Female_Only_Movie,
                        TM.Total_Movies AS Total_Movies
                     FROM   TOTAL_MOVIES TM JOIN  FEMALE_ONLY_MOVIES FOM
                     ON (TM.YEAR = FOM.YEAR)   """
grader_5b(query5b)
```

```
   year  Percentage_Female_Only_Movie  Total_Movies
0  1939                      0.500000             2
1  1999                      0.015152            66
2  2000                      0.015625            64
3  2018                      0.009615           104
CPU times: user 396 ms, sys: 9.88 ms, total: 406 ms
Wall time: 406 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 [15]:

```
%%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, MC.count
               FROM  Movie M
               JOIN (SELECT mid, COUNT(DISTINCT pid) count
                       FROM M_Cast
                   GROUP BY mid) MC
                ON(MC.mid = M.mid)
                ORDER BY MC.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 76.9 ms, sys: 7.95 ms, total: 84.8 ms
Wall time: 86.4 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 [16]:

```
%%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 = """ SELECT CAST(SUBSTR(TRIM(M.year),-4)AS INTEGER) AS Movie_Year, COUNT(*) AS
Total_Movies
                FROM Movie M
                GROUP BY CAST(SUBSTR(TRIM(M.year),-4)AS INTEGER) """
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 11.7 ms, sys: 277 µs, total: 12 ms
Wall time: 12.8 ms
```

In [17]:

```
%%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 YEAR_MOVIES AS (
             SELECT CAST(SUBSTR(TRIM(M.year),-4)AS INTEGER) AS Movie_Year,
                    COUNT(*) AS Total_Movies
               FROM Movie M
             GROUP BY CAST(SUBSTR(TRIM(M.year),-4)AS INTEGER))
             SELECT  *
             FROM YEAR_MOVIES YM JOIN YEAR_MOVIES YM1
             ON (YM1.Movie_Year >= YM.movie_year AND YM1.movie_year <= YM.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 13.4 ms, sys: 32 µs, total: 13.4 ms
Wall time: 13.7 ms
```

In [18]:

```
%%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 = """ WITH YEAR_MOVIES AS (
                SELECT CAST(SUBSTR(TRIM(M.year),-4)AS INTEGER) AS Movie_Year,
                       COUNT(*) AS Total_Movies
                  FROM Movie M
                GROUP BY CAST(SUBSTR(TRIM(M.year),-4)AS INTEGER))
             SELECT  SUM(YM1.total_movies) AS Decade_Movie_Count,
                     GROUP_CONCAT(YM1.movie_year,'-') AS Decade
               FROM YEAR_MOVIES YM JOIN YEAR_MOVIES YM1
               ON (YM1.Movie_Year >= YM.movie_year AND YM1.movie_year <= YM.movie_year
 + 9)
              GROUP BY YM.movie_year
              ORDER BY Decade_Movie_Count DESC
              LIMIT 1     """
grader_7(query7)
# if you check the output we are printinng all the year in that decade, its fine you ca
n print 2008 or 2008-2017
```

```
   Decade_Movie_Count                                          Decade
0                1203  2008-2009-2010-2011-2012-2013-2014-2015-2016-2017
CPU times: user 12.1 ms, sys: 0 ns, total: 12.1 ms
Wall time: 12.5 ms
```

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

In [19]:

```
%%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, MD.pid director, COUNT(MD.mid)
                FROM  M_Director MD, M_Cast MC ON (MD.MID = MC.MID)
            GROUP BY MC.pid, MD.pid"""
grader_8a(query8a)

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

```
        actor   director  COUNT(MD.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 322 ms, sys: 22.8 ms, total: 345 ms
Wall time: 349 ms
```

In [20]:

```
%%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
                 YASH_CHOPRA AS
                 (
                    SELECT P.pid
                      FROM Person P
                     WHERE TRIM(P.name) ='Yash Chopra'
                 ),
                 ACTOR_DIR AS
                 (
                    SELECT  MC.pid actor, MD.pid director, COUNT(MD.mid) as count
                      FROM  M_Director MD, M_Cast MC ON (MD.MID = MC.MID)
                    GROUP BY MC.pid, MD.pid
                 ),
                 ACTOR_COUNTS AS
                 (
                   SELECT AD.actor, AD.count yash_counts, max(AD1.count) other_dir_coun
ts
                     FROM  YASH_CHOPRA YC JOIN ACTOR_DIR AD ON (YC.pid = AD.director)
                     JOIN  ACTOR_DIR AD1 ON (AD.actor = AD1.actor )
                 GROUP BY AD.actor, AD.count
                 )
                 SELECT P2.name,AC.yash_counts as count
                   FROM ACTOR_COUNTS AC JOIN Person P2 On (TRIM(AC.actor) = P2.pid)
                  WHERE yash_counts >= other_dir_counts
                ORDER BY AC.yash_counts desc
                """
grader_8(query8)
```

```
                Name  count
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 1.49 s, sys: 36.5 ms, total: 1.52 s
Wall time: 1.54 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 [21]:

```
%%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
                S1_MOVIES AS
                (
                SELECT DISTINCT MC.pid, MC.mid
                  FROM Person P JOIN M_CAST MC ON (P.pid = TRIM(MC.pid))
                 WHERE TRIM(P.name) = 'Shah Rukh Khan'
                ) ,
                S1_ACTORS AS
                (
                SELECT DISTINCT MC.pid
                  FROM  M_Cast MC JOIN S1_MOVIES S1M
                  ON (MC.mid = S1M.mid AND MC.pid <> S1M.pid)
                )
               SELECT pid FROM S1_ACTORS
               """
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 ac
tors
# removing S1 actors from the combined list of S1 & S2 actors, so that we get only S2 a
ctors
```

```
          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 162 ms, sys: 5.94 ms, total: 167 ms
Wall time: 170 ms
```

In [22]:

```
%%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
               S1_MOVIES AS
               (
                SELECT DISTINCT MC.pid, MC.mid
                  FROM Person P JOIN M_CAST MC ON (P.pid = TRIM(MC.pid))
                 WHERE TRIM(P.name) = 'Shah Rukh Khan'
               ) ,
               S1_ACTORS AS
               (
                SELECT DISTINCT MC.pid
                  FROM  M_Cast MC JOIN S1_MOVIES S1M
                  ON (MC.MID = S1M.MID AND TRIM(MC.PID) <> TRIM(S1M.PID))
                ) ,
                S2_MOVIES AS
                (
                SELECT DISTINCT MC.mid
                  FROM M_Cast MC JOIN S1_ACTORS S1A
                  ON (MC.PID = S1A.PID )
                ) ,
                S2_ACTORS AS
                (
                SELECT DISTINCT MC.pid
                  FROM M_Cast MC JOIN S2_MOVIES S2M
                  ON (MC.mid = S2M.mid )
                ) ,
                ONLY_S2_ACTORS AS
                (
                SELECT DISTINCT S2A.pid
                  FROM S2_ACTORS S2A
                 WHERE  NOT EXISTS
                  (SELECT '1'
                     FROM S1_ACTORS S1A
                    WHERE S1A.pid = S2A.pid
                   )
                )
                SELECT  P.Name
                  FROM  ONLY_S2_ACTORS S2O Join Person P
                  ON (TRIM(S2O.pid) = P.pid)
                 WHERE S2O.pid NOT IN (SELECT pid FROM S1_MOVIES)
                 """
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 746 ms, sys: 14.8 ms, total: 761 ms
Wall time: 764 ms
```