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("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()
```

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))
#     print(q1_results.shape)
    assert (q1_results.shape == (232,3))

query1 = """SELECT p.Name, m.title, m.year
            FROM Movie m
            JOIN M_Director mD
                ON mD.MID = m.MID
            JOIN Person p
                ON mD.PID = p.PID
            WHERE m.MID IN
                (SELECT MID FROM M_Genre
                    WHERE GID IN
                        (SELECT GID FROM Genre
                            WHERE 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 64.6 ms, sys: 11.9 ms, total: 76.5 ms
Wall time: 71.9 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 Name Actor_Name FROM Person p
         WHERE PID IN (
             SELECT TRIM(PID) FROM M_Cast
         WHERE M_Cast.MID IN
             (SELECT TRIM(MID) FROM Movie
         WHERE title ='Anand'))
         """

grader_2(query2)
```

```
          Actor_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 16 ms, sys: 0 ns, total: 16 ms
Wall time: 14.6 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 169 ms, sys: 4.19 ms, total: 173 ms
Wall time: 170 ms
```

In [9]:

```
%%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 TRIM(p.Name) Actor_Name FROM
                (SELECT DISTINCT TRIM(MC.PID) PID FROM Movie M
                JOIN M_Cast MC ON M.MID = MC.MID
                WHERE CAST(SUBSTR(M.year,-4) AS Integer) < 1970) l_1970
            JOIN
                (SELECT DISTINCT TRIM(MC.PID) PID FROM Movie M
                JOIN M_Cast MC ON M.MID = MC.MID
                WHERE CAST(SUBSTR(M.year,-4) AS Integer) > 1990) g_1990
            ON l_1970.PID = g_1990.PID
            JOIN Person P
            ON l_1970.PID = TRIM(P.PID)
            """

# https://youtu.be/_SanZ41uTlw
# https://youtu.be/vLjAG9eXkcU

query3 = """
         WITH
         before_1970 AS (
         SELECT DISTINCT TRIM(MC.PID) PID FROM Movie M
             JOIN M_Cast MC ON M.MID = MC.MID
             WHERE CAST(SUBSTR(M.year,-4) AS Integer) < 1970),

         after_1990 AS (
         SELECT DISTINCT TRIM(MC.PID) PID FROM Movie M
             JOIN M_Cast MC ON M.MID = MC.MID
             WHERE CAST(SUBSTR(M.year,-4) AS Integer) > 1990)

         SELECT DISTINCT TRIM(p.Name) Actor_Name FROM before_1970
         JOIN after_1990
         ON before_1970.PID = after_1990.PID
         JOIN Person P
         ON before_1970.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
CPU times: user 201 ms, sys: 9.14 ms, total: 210 ms
Wall time: 208 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))
    return (query_4a.shape == (1462,2))

query_4a = """
           SELECT PID, COUNT(MID) Movie_Cout FROM M_Director
           GROUP BY PID
           """

print(grader_4a(query_4a))

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

```
        PID  Movie_Cout
0  nm0000180           1
1  nm0000187           1
2  nm0000229           1
3  nm0000269           1
4  nm0000386           1
5  nm0000487           2
```

```
6  nm0000965           1
7  nm0001060           1
8  nm0001162           1
9  nm0001241           1
True
CPU times: user 5.55 ms, sys: 403 µs, total: 5.95 ms
Wall time: 4.51 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
         dir_mov_count AS (
         SELECT PID, COUNT(MID) AS Movie_Cout FROM M_Director
         GROUP BY PID HAVING Movie_Cout >= 10)

         SELECT DISTINCT(p.Name) Director_Name, dmc.Movie_Cout Movie_Count
         FROM dir_mov_count dmc
         JOIN Person p
         ON p.PID = dmc.PID
         ORDER BY dmc.Movie_Cout DESC
         """

grader_4(query4)
```

```
          Director_Name  Movie_Count
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 11.4 ms, sys: 0 ns, total: 11.4 ms
Wall time: 10.2 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 = """
            WITH
            mc_m AS
            (SELECT * FROM M_Cast mc
            WHERE MID IN
            (SELECT TRIM(MID) FROM Movie))

            SELECT MID, Gender, COUNT(*) AS Count FROM Person p
            JOIN mc_m
            ON
            TRIM(mc_m.PID) = p.PID
            GROUP BY MID, Gender
            """

print(grader_5aa(query_5aa))
print('\n','=' * 27, '\n')

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 = """
            WITH
            mc_m AS
            (SELECT * FROM M_Cast mc
            WHERE MID IN
            (SELECT TRIM(MID) FROM Movie))

            SELECT MID, Gender, COUNT(*) AS Count FROM Person p
            JOIN mc_m
```

```
            ON
            TRIM(mc_m.PID) = p.PID
            WHERE Gender = 'Male'
            GROUP BY MID, Gender
            """

print(grader_5ab(query_5ab))

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

```
         MID  Gender  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 Gender  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 252 ms, sys: 9.69 ms, total: 261 ms
Wall time: 259 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 = """
          WITH
          male_nd_none AS
          (SELECT DISTINCT(TRIM(mc.MID)) MID FROM M_Cast mc
          JOIN Person p
          ON TRIM(mc.PID) = p.PID
          WHERE
          TRIM(p.Gender) = 'Male' OR 'None')

          SELECT CAST(SUBSTR(year,-4) AS Integer) Year, COUNT(*) Female_Cast_Only_Movies FROM Movie
          WHERE MID NOT IN male_nd_none
          GROUP BY Year
          ORDER BY Year
          """

grader_5a(query5a)
```

```
   Year  Female_Cast_Only_Movies
0  1939                        1
1  1999                        1
2  2000                        1
3  2018                        1
CPU times: user 99.5 ms, sys: 0 ns, total: 99.5 ms
Wall time: 98.1 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))
    print(q5b_results.shape)
    assert (q5b_results.shape == (4,3))
```

```
# Reference
'''
https://social.msdn.microsoft.com/Forums/sqlserver/en-US/4cf91f5a
-9c22-451a-8811-dc6621fe365f/how-to-get-values-with-decimals-when-dividing-2-integers-in-tsql?
forum=sqlintegrationservices
'''
query5b = """
   WITH
   male_nd_none AS
   (SELECT DISTINCT(TRIM(mc.MID)) MID FROM M_Cast mc
   JOIN Person p
   ON TRIM(mc.PID) = p.PID
   WHERE
   TRIM(p.Gender) = 'Male' OR 'None'),

   year_female AS
   (SELECT CAST(SUBSTR(year,-4) AS Integer) Year, COUNT(*) Female_Cast_Only
   FROM Movie
   WHERE MID NOT IN male_nd_none
   GROUP BY Year),

   total_movie AS
   (SELECT year, COUNT(*) Total_Movies FROM Movie
   WHERE CAST(SUBSTR(year,-4) AS Integer)
   IN
   (SELECT Year FROM year_female)
   GROUP BY CAST(SUBSTR(year,-4) AS Integer))

   SELECT tm.year Year,
   CAST(ym.Female_Cast_Only AS FLOAT) / CAST (tm.Total_Movies AS FLOAT) Percentage_Female_Only_Movie,
   tm.total_movies
   FROM total_movie tm
   JOIN year_female ym
   ON ym.year = tm.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
(4, 3)
CPU times: user 100 ms, sys: 5.65 ms, total: 106 ms
Wall time: 103 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 = """
         WITH

         Cast_Count AS (
         SELECT MID,COUNT(DISTINCT(PID)) Cast_Count FROM M_Cast
         GROUP BY MID)

         SELECT m.title, cc.Cast_Count FROM Movie m
         JOIN Cast_Count cc
         ON m.MID = cc.MID
         ORDER BY cc.Cast_Count DESC
         """

grader_6(query6)
```

```
                       title  Cast_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 50.6 ms, sys: 1.2 ms, total: 51.8 ms
Wall time: 50.1 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 year AS Movie_Year, COUNT(*) AS Total_Movies FROM Movie
          GROUP BY CAST(SUBSTR(TRIM(year),-4) AS INTEGER)
          """

grader_7a(query7a)
```

```
  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 4.2 ms, sys: 79 µs, total: 4.28 ms
Wall time: 3.49 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))

"""
    ***
    Write a query that will do joining of the above table(7a) with itself
    such that you will join with only rows if the second tables year is <= current_year+9
    and more than or equal current_year
    ***
"""

# select year, count(*) as total_movies from Movie group by year
# select year, count(*) as total_movies from Movie group by year
# select step1.year, step1.totalmovies, step2.year, step2.totalmovies FROM step1
# join step2 where step2.year <= step1.year+9 AND step2.year >= step1.year

query7b = """
          WITH
          year_total AS
          (SELECT year AS Movie_Year, COUNT(*) AS Total_Movies FROM Movie
          GROUP BY year)

          SELECT * FROM year_total yt_1
          JOIN year_total yt_2
          ON (yt_2.Movie_Year <= yt_1.Movie_Year+9  AND
                yt_2.Movie_Year >= yt_1.Movie_Year)
          """

query7b = """
          WITH
          year_total AS
          (SELECT year AS Movie_Year, COUNT(*) AS Total_Movies FROM Movie
          GROUP BY year)

          SELECT * FROM year_total yt_1
          JOIN year_total yt_2
          WHERE (yt_2.Movie_Year <= yt_1.Movie_Year+9  AND
                yt_2.Movie_Year >= yt_1.Movie_Year)
          """

grader_7b(query7b)
```

```
  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 7.17 ms, sys: 0 ns, total: 7.17 ms
Wall time: 6.2 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))

query7A = """
          WITH
          year_ AS
          (SELECT DISTINCT(year) year FROM Movie)

          SELECT COUNT(*) AS Decade_Movie_Count, yt.year AS Decade FROM year_ yt
          JOIN Movie m
          ON (CAST(SUBSTR(m.year,-4) AS Integer) <= yt.year+9  AND
                  CAST(SUBSTR(m.year,-4) AS Integer) >= Decade)
          GROUP BY yt.year+9
          ORDER BY COUNT(*) DESC
          LIMIT 1
          """

grader_7(query7A)
# if you check the output we are printinng all the year in that decade,
# its fine you can print 2008 or 2008-2017
```

```
   Decade_Movie_Count Decade
0                1203   2008
CPU times: user 83.5 ms, sys: 0 ns, total: 83.5 ms
Wall time: 82.3 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 AS actor, md.PID AS director, COUNT(*) AS movies FROM M_Cast mc
          JOIN M_Director md
          ON md.MID = TRIM(mc.MID)
          GROUP BY actor, director
          ORDER BY actor, director
          """
grader_8a(query8a)
```

```
       actor   director  movies
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 169 ms, sys: 7.83 ms, total: 177 ms
Wall time: 175 ms
```

In [20]:

```
%%time

# https://stackoverflow.com/a/69602577

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_count AS (
            SELECT DISTINCT mc.PID AS actor, COUNT(*) AS movies FROM M_Cast mc
            JOIN M_Director md
            ON md.MID = mc.MID
            WHERE md.PID IN (SELECT PID FROM Person WHERE Name LIKE '%Yash Chopra%')
            GROUP BY actor, md.PID
            ORDER BY movies DESC),

            except_yash AS (
            SELECT actor AS actor, MAX(movies) AS movies FROM (
            SELECT mc.PID AS actor, COUNT(*) AS movies FROM M_Cast mc
            JOIN M_Director md
            ON md.MID = mc.MID
            WHERE md.PID NOT IN (SELECT PID FROM Person WHERE Name LIKE '%Yash Chopra%')
            GROUP BY actor, md.PID
            ORDER BY movies DESC)
            GROUP BY actor
            ORDER BY movies DESC),

            final AS (
            SELECT yc.actor, yc.movies FROM yash_count yc
            LEFT JOIN except_yash ey
            ON yc.actor = ey.actor
            WHERE (yc.movies >= ey.movies OR ey.movies IS NULL))

            SELECT p.Name, f.movies FROM final f
            JOIN Person p
            ON TRIM(f.actor) = p.PID
         """

grader_8(query8)
```

```
                Name  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 318 ms, sys: 10.2 ms, total: 328 ms
Wall time: 326 ms
```

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))
    assert (q9a_results.shape == (2382, 1))

query9a = """
          WITH

          srk_PID AS (
          SELECT TRIM(PID) PID_SRK FROM Person
          WHERE TRIM(Name) LIKE 'Shah Rukh Khan'),

          srk_movie_list AS (
          SELECT TRIM(MID) SRK_MovieID FROM M_Cast mc, srk_PID
          WHERE TRIM(PID) = srk_PID.PID_SRK),

          full_list AS (
          SELECT DISTINCT TRIM(mc.PID) S1_PID FROM M_Cast mc
          WHERE mc.MID IN (
          SELECT * FROM srk_movie_list))

          SELECT S1_PID FROM full_list fl, srk_PID
          WHERE  S1_PID <> srk_PID.PID_SRK
          """
```

```
grader_9a(query9a)
```

```
      S1_PID
0  nm0004418
1  nm1995953
2  nm2778261
3  nm0631373
4  nm0241935
5  nm0792116
6  nm1300111
7  nm0196375
8  nm1464837
9  nm2868019
CPU times: user 49.6 ms, sys: 443 µs, total: 50 ms
Wall time: 47.9 ms
```

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 [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))

# 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 actors
# removing S1 actors from the combined list of S1 & S2 actors, so that we get only S2 actors

query9 = """
          WITH

          srk_PID AS (
          SELECT TRIM(PID) PID_SRK FROM Person
          WHERE TRIM(Name) LIKE 'Shah Rukh Khan'),

          srk_movie_list AS (
          SELECT TRIM(MID) SRK_MovieID FROM M_Cast mc, srk_PID
          WHERE TRIM(PID) = srk_PID.PID_SRK),

          full_list AS (
          SELECT DISTINCT TRIM(mc.PID) S1_PID FROM M_Cast mc
          WHERE mc.MID IN (
          SELECT * FROM srk_movie_list)),

          S1_Actors AS (
          SELECT S1_PID FROM full_list fl, srk_PID
          WHERE  S1_PID <> srk_PID.PID_SRK),

          S2_Movies AS(
          SELECT DISTINCT TRIM(mc.MID) S2_MID FROM M_Cast mc, S1_Actors s1A
          WHERE TRIM(mc.PID) = s1A.S1_PID
          ),

          S2_Actors AS (
          SELECT DISTINCT TRIM(mc.PID) S2_PID FROM M_Cast mc, S2_Movies s2m
          WHERE s2m.S2_MID = mc.MID),

          S2_Actors_with_srk AS (
          SELECT DISTINCT s2A.S2_PID FROM S2_Actors s2A
          WHERE s2A.S2_PID NOT IN (SELECT * FROM S1_Actors)),

          F_S2_Actors AS (
          SELECT S2_PID FROM S2_Actors_with_srk, srk_PID WHERE S2_PID <> srk_PID.PID_SRK)

          SELECT p.Name FROM Person p, F_S2_Actors s2
          WHERE TRIM(p.PID) = s2.S2_PID
          """

grader_9(query9)
```

```
                  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 285 ms, sys: 11.4 ms, total: 297 ms
Wall time: 295 ms
```