SQL Assignment

In [1]:

```python
import pandas as pd
import sqlite3

from IPython.display import display, HTML
```

In [2]:

```python
# 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-L1DdNxEK606nG2jS31MbrMh-OnXM/view?usp=sharing
```

In [3]:

```python
conn = sqlite3.connect("Db-IMDB-Assignment.db")
```

Overview of all tables

In [4]:

```python
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]:

```python
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]:

```python
%%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 = """ WITH leap_year_comedy_movie_id AS (
                 SELECT TRIM(MID) AS movie_id
                 FROM Movie m
                 WHERE CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER)%4==0 AND MID IN(
                     SELECT TRIM(MID)
                     FROM M_Genre
                     WHERE GID IN(
                         SELECT GID
                         FROM Genre
                         WHERE TRIM(Name) LIKE '%Comedy%' ))),


             movie_name_year_id AS(
                SELECT title,year, MID
                FROM Movie m
                WHERE MID IN(
                    SELECT movie_id
                    FROM leap_year_comedy_movie_id ))


            SELECT name, title, year
            FROM Person AS p
            INNER JOIN
            M_Director AS m_d
            ON p.PID = m_d.PID
            INNER JOIN
            movie_name_year_id AS m_n_y_id
            ON m_d.MID = m_n_y_id.MID """



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
Wall time: 80 ms
```

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

In [7]:

```python
%%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
            FROM Person
            WHERE TRIM(PID) IN(
                SELECT TRIM(PID)
                FROM M_Cast
                WHERE TRIM(MID) IN(
                    SELECT TRIM(MID)
                    FROM Movie
                    Where TRIM(title) == 'Anand' ))"""
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
Wall time: 33 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]:

```python
%%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
Wall time: 333 ms
```

In [9]:

```python
%%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 = """WITH before_1970 AS (
                SELECT PID AS pid_1970
                FROM M_Cast
                WHERE MID IN(
                SELECT MID
                FROM Movie
                WHERE CAST(SUBSTR(year,-4) AS Integer)<1970)),
            after_1990 AS (
                SELECT PID AS pid_1990
                FROM M_Cast
                WHERE MID IN(
                SELECT MID
                FROM Movie
                WHERE CAST(SUBSTR(year,-4) AS Integer)>1990)),

            person_id (pids) AS (SELECT TRIM(pid_1970)
            FROM before_1970
            INTERSECT
            SELECT TRIM(pid_1990)
            FROM after_1990 )


            SELECT Name
            FROM Person p
            WHERE p.pid IN(
            SELECT pids
            FROM person_id)

          """
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
Wall time: 160 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]:

```python
%%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))

#*** Write a query, which will return all the directors(id's) along with the number of movies they directed ***

query_4a ="""SELECT name, COUNT(name) AS movie_count
             FROM Person AS p
             INNER JOIN
             M_Director AS m_d
             ON p.pid = m_d.pid
             GROUP BY m_d.pid
             ORDER BY movie_count DESC"""



print(grader_4a(query_4a))

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

```
                   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
True
Wall time: 35 ms
```

In [11]:

```python
%%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, COUNT(name) AS movie_count
             FROM Person AS p
             INNER JOIN
             M_Director AS m_d
             ON p.pid = m_d.pid
             GROUP BY m_d.pid
             HAVING movie_count >= 10
             ORDER BY movie_count DESC"""
grader_4(query4)
```

```
                   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
Wall time: 33 ms
```

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

In [12]:

```python
%%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))

# *** Write your query that will get movie id, and number of people for each geneder ***


query_5aa ="""SELECT m_c.MID, Gender, COUNT(*)
              FROM M_Cast AS m_c
              INNER JOIN
              Person AS p
              ON TRIM(m_c.PID) = p.PID
              GROUP BY m_c.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))

# *** Write your query that will have at least one male actor try to use query that you have written above ***

query_5ab ="""SELECT m_c.MID, Gender, COUNT(*)
              FROM M_Cast AS m_c
              INNER JOIN
              Person AS p
              ON TRIM(m_c.PID) = p.PID
              GROUP BY m_c.MID, p.Gender
              HAVING p.Gender == 'Male'"""

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
Wall time: 405 ms
```

In [13]:

```python
%%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_present_mid AS(
              SELECT m_c.MID
              FROM M_Cast AS m_c
              INNER JOIN
              Person AS p
              ON TRIM(m_c.PID) = p.PID
              WHERE p.Gender == 'Male'),

              Female_only_mid AS(
              SELECT MID
              FROM Movie m
              WHERE MID NOT IN(
              SELECT MID
              FROM Male_present_mid AS m_p_m))

              SELECT CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER) year, COUNT(m.MID) Number_of_movies
              FROM movie AS m
              INNER JOIN
              Female_only_mid AS f_o_m
              ON m.MID = f_o_m.MID
              GROUP BY m.year"""



grader_5a(query5a)
```

```
   year  Number_of_movies
0  1939                 1
1  1999                 1
2  2000                 1
3  2018                 1
Wall time: 192 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]:

```python
%%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 movies_each_year AS(
              SELECT CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER) year, COUNT(year) Total_Movies
              FROM Movie m
              GROUP BY CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER)),

              Male_present_mid AS(
              SELECT m_c.MID
              FROM M_Cast AS m_c
              INNER JOIN
              Person AS p
              ON TRIM(m_c.PID) = p.PID
              WHERE p.Gender == 'Male'),

              Female_only_mid AS(
              SELECT MID
              FROM Movie m
              WHERE MID NOT IN(
              SELECT MID
              FROM Male_present_mid AS m_p_m)),

              Female_only_movie_year AS(SELECT CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER) year, COUNT(m.MID) Numb
              FROM movie AS m
              INNER JOIN
              Female_only_mid AS f_o_m
              ON m.MID = f_o_m.MID
              GROUP BY m.year)

              SELECT CAST(SUBSTR(TRIM(f_o_m_y.year),-4) AS INTEGER) year, CAST(f_o_m_y.Number_of_movies AS FLOA
              /m_e_y.Total_Movies Percentage_female_only_movies, m_e_y.Total_Movies
              FROM Female_only_movie_year AS f_o_m_y
              INNER JOIN
              movies_each_year AS m_e_y
              ON f_o_m_y.year = m_e_y.year
              GROUP BY f_o_m_y.year

              """



grader_5b(query5b)
```

```
   year  Percentage_female_only_movies  Total_Movies
0  1939                       0.500000             2
1  1999                       0.015152            66
2  2000                       0.015625            64
3  2018                       0.009615           104
Wall time: 198 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]:

```python
%%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(m_c.PID)) cast_count
            FROM Movie m
            INNER JOIN
            M_Cast m_c
            ON m_c.MID = m.MID
            GROUP BY m_c.MID
            ORDER BY 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
Wall time: 249 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]:

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

#*** Write a query that computes number of movies in each year ***

query7a = """SELECT year , COUNT(MID)
             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
```

```
   year  COUNT(MID)
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
Wall time: 5.99 ms
```

In [17]:

```python
%%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 equ


query7b = """WITH cast_count_year AS(
             SELECT CAST(SUBSTR(TRIM(year),-4) AS INTEGER) year, COUNT(MID) Total_movies
             FROM Movie m
             GROUP BY CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER))

             SELECT ccy_1.year, ccy_1.Total_movies, ccy_2.year, ccy_2.Total_movies
             FROM cast_count_year ccy_1
             INNER JOIN
             cast_count_year ccy_2
             WHERE  ccy_2.year <= ccy_1.year+9 AND ccy_2.year >= ccy_1.year



          """
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
```

```
   year  Total_movies  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
Wall time: 9.01 ms
```

In [18]:

```python
%%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 cast_count_year AS(
             SELECT CAST(SUBSTR(TRIM(year),-4) AS INTEGER) year, COUNT(MID) Total_movies
             FROM Movie m
             GROUP BY CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER))

             SELECT  SUM(ccy_2.Total_movies) Decade_movie_count, GROUP_CONCAT(ccy_2.year) Decade
             FROM cast_count_year ccy_1
             INNER JOIN
             cast_count_year ccy_2
             WHERE  ccy_2.year <= ccy_1.year+9 AND ccy_2.year >= ccy_1.year
             GROUP BY ccy_1.year
             ORDER BY SUM(ccy_2.Total_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 2008-20
```

```
   Decade_movie_count                                             Decade
0                1203  2008,2009,2010,2011,2012,2013,2014,2015,2016,2017
Wall time: 8 ms
```

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

In [19]:

```python
%%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  m_c.PID Actor_ID, m_d.PID Director_ID, COUNT(m_c.MID) Movies
              FROM M_Cast m_c
              INNER JOIN
              M_Director m_d
              ON m_c.MID = m_d.MID
              GROUP BY m_c.PID,m_d.PID
              """
grader_8a(query8a)

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

```
    Actor_ID Director_ID  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
Wall time: 335 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 actor_director_id_movie_count AS(
              SELECT  m_c.PID Actor_ID, m_d.PID Director_ID, COUNT(m_c.MID) Movies
              FROM M_Cast m_c
              INNER JOIN
              M_Director m_d
              ON m_c.MID = m_d.MID
              GROUP BY m_c.PID,m_d.PID),

              yash_chopra_actor_movies AS(
              SELECT Actor_ID, Movies, Director_ID
              FROM actor_director_id_movie_count adimc
              INNER JOIN
              Person p
              ON p.PID = adimc.Director_ID
              WHERE TRIM(p.name) = 'Yash Chopra'),

              not_yash_chopra_actor_movies AS(
              SELECT adimc.Actor_ID, adimc.Director_ID, adimc.Movies
              FROM actor_director_id_movie_count adimc
              INNER JOIN
              yash_chopra_actor_movies ycam
              ON adimc.Actor_ID = ycam.Actor_ID),

              max_movies_not_yash_chopra AS(
              SELECT Actor_ID, MAX(Movies) max_movies
              FROM not_yash_chopra_actor_movies nycam
              GROUP BY  nycam.Actor_ID),

              actor_id_movies AS(
              SELECT TRIM(ycam.Actor_ID) Actor_ID, ycam.Movies
              FROM yash_chopra_actor_movies ycam
              INNER JOIN
              max_movies_not_yash_chopra mmoyc
              ON ycam.Actor_ID = mmoyc.Actor_ID
              WHERE ycam.Movies = mmoyc.max_movies
              ORDER BY ycam.Movies DESC)

              SELECT p.name, aim.Movies
              FROM Person p
              INNER JOIN
              actor_id_movies aim
              ON p.PID = aim.Actor_ID
              ORDER BY Movies DESC

              """

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)
Wall time: 658 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))
    print(q9a_results.shape)
    assert (q9a_results.shape == (2382, 1))

query9a = """
             WITH srk_id AS(
             SELECT TRIM(PID) PID , Name
             FROM Person p
             WHERE TRIM(Name) == 'Shah Rukh Khan'),

             srk_movies AS(
             SELECT MID
             FROM M_cast m_c
             WHERE TRIM(m_c.PID) IN(
             SELECT srk_id.PID
             FROM srk_id)),


             s1_with_srk AS (
             SELECT DISTINCT PID
             FROM M_cast m_c
             WHERE TRIM(MID) IN(
             SELECT TRIM(MID)
             FROM srk_movies))


             SELECT s1_s.PID
             FROM s1_with_srk s1_s
             WHERE TRIM(PID) NOT IN(
             SELECT PID
             FROM srk_id )



             """


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

```
         PID
0  nm0004418
1  nm1995953
2  nm2778261
3  nm0631373
4  nm0241935
5  nm0792116
6  nm1300111
7  nm0196375
8  nm1464837
9  nm2868019
(2382, 1)
Wall time: 60.7 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 srk_id AS(
             SELECT TRIM(PID) PID , Name
             FROM Person p
             WHERE TRIM(Name) == 'Shah Rukh Khan'),

             srk_movies AS(
             SELECT MID
             FROM M_cast m_c
             WHERE TRIM(m_c.PID) IN(
             SELECT srk_id.PID
             FROM srk_id)),

             s1_with_srk AS (
             SELECT DISTINCT PID
             FROM M_cast m_c
             WHERE TRIM(MID) IN(
             SELECT TRIM(MID)
             FROM srk_movies)),

             S1 AS (
             SELECT s1_s.PID
             FROM s1_with_srk s1_s
             WHERE TRIM(PID) NOT IN(
             SELECT PID
             FROM srk_id )),

             s2_movies AS (SELECT DISTINCT MID
             FROM M_cast m_c
             WHERE PID IN(
             SELECT PID
             FROM s1)),

             s1_s2_movie_actors AS (
             SELECT DISTINCT PID
             FROM M_cast m_c
             WHERE MID IN(
             SELECT MID
             FROM s2_movies)),

             S2_ID AS (
             SELECT PID
             FROM s1_s2_movie_actors s1_s2
             WHERE PID NOT IN(
             SELECT PID
             FROM s1_with_srk))

             SELECT Name
             FROM Person p
             WHERE PID IN(
             SELECT TRIM(PID)
             FROM S2_ID)
             """
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)
Wall time: 355 ms
```