SQL Assignment

In [10]:

```
conn = sqlite3.connect("C:/Users/asus/Documents/AAIC_assign/3_SQL/Db-IMDB-Assignment.db")
cursor = conn.cursor()
```

Overview of all tables

In [11]:

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

In [12]:

```
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 [13]:

```
#This query is to check what type of corrupt values do we have under year column
q_val="""SELECT distinct Year,conv_year from (SELECT *,cast(year as int) as conv_year f
rom Movie)
WHERE conv_year=0"""
res=pd.read_sql_query(q_val,conn)
print(res)
```

```
Empty DataFrame
Columns: [year, conv_year]
Index: []
```

In [14]:

```
m1='UPDATE Movie SET year=replace(Year,"I","")'
m2='UPDATE Movie SET Year=replace(Year,"XV","")'
m3='UPDATE Movie SET Year=replace(Year,"V","")'
m4='UPDATE Movie SET Year=ltrim(Year)'

cursor.execute(m1)
cursor.execute(m2)
cursor.execute(m3)
cursor.execute(m4)
```

Out[14]:

```
<sqlite3.Cursor at 0x13aa3676dc0>
```

In [15]:

```
%%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 Name as DirectorName,title as MovieName,year from (SELECT RR.MID,ti
tle,year,PID from (SELECT MID,title,year,G.GID from (SELECT M.MID,title,year,MG.GID fro
m Movie as M
inner join M_Genre as MG
on M.MID=MG.MID
where M.year%4=0 or M.year%100=0 or M.year%400=0) as tab
inner join Genre G
on tab.GID=G.GID and G.Name like '%Comedy%') as RR
inner join M_Director MD
on RR.MID=MD.MID) as fill
inner join Person as P
on fill.PID=P.PID """
grader_1(query1)
```

```
        DirectorName                          MovieName  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: 307 ms
```

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

In [16]:

```
%%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 (SELECT ltrim(mc.PID) as PID from Movie M
INNER JOIN M_Cast MC
on M.MID=MC.MID and M.title='Anand') as fill
inner join Person P
on fill.PID=P.PID """
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
Wall time: 850 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 [17]:

```
%%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: 2.28 s
```

In [18]:

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

q1="DROP TABLE IF EXISTS temp.temp_less_1970;"
q2="DROP TABLE IF EXISTS temp.temp_after_1970;"
q_less_1970="""CREATE TEMP table temp_less_1970 as
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;"""

q_more_1970="""CREATE temp table temp_after_1970 as
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 ;"""
cursor.execute(q1)
cursor.execute(q2)
cursor.execute(q_less_1970)
cursor.execute(q_more_1970)

query3 = """
SELECT DISTINCT PP.Name from (
SELECT P.PID,P.Name from Person as P
inner join temp_less_1970 as t1
on P.PID=t1.PID) as PP
inner join temp_after_1970 t2
on PP.PID=t2.PID
"""
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: 3.5 s
```

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 [19]:

```
%%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 tt.PID as Director_ID,count(tt.MID) as Movie_count from (SELECT M
D.PID,M.MID from Movie M
inner join M_Director MD
on M.MID=MD.MID) as tt
GROUP by tt.PID """
print(grader_4a(query_4a))

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

```
  Director_ID  Movie_count
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
Wall time: 45.7 ms
```

In [20]:

```
%%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 as Director_Name,count(title) as Movie_count from (SELECT Name,
M.title from (SELECT P.PID as PID,P.Name as Name,MD.MID  from Person P
INNER join M_Director MD
on P.PID=MD.PID) as Dir_Name
inner JOIN Movie M
on Dir_Name.MID=M.MID) as final_list
GROUP by Name
HAVING count(title)>=10"""
grader_4(query4)
```

```
              Director_Name  Movie_count
0   Abbas Alibhai Burmawalla           17
1   Ananth Narayan Mahadevan           13
2               Anees Bazmee           12
3                Anil Sharma           12
4             Anurag Kashyap           13
5            Basu Chatterjee           19
6                  Bimal Roy           10
7               David Dhawan           39
8                  Dev Anand           13
9            Govind Nihalani           11
Wall time: 106 ms
```

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

In [21]:

```
%%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 fin_data.MID,fin_data.Gender,count(Gender)as Count  from (SELECT
 M.MID,person_data.Gender from (SELECT MC.MID,P.Gender from M_Cast MC
inner JOIN Person P
on ltrim(MC.PID)=P.PID) as person_data
inner join Movie M
on person_data.MID=M.MID) fin_data
GROUP by fin_data.MID,fin_data.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 fin_data.MID,fin_data.Gender,count(Gender)as Count  from (SELECT
 M.MID,person_data.Gender from (SELECT MC.MID,P.Gender from M_Cast MC
inner JOIN Person P
on ltrim(MC.PID)=P.PID) as person_data
inner join Movie M
on person_data.MID=M.MID) fin_data
GROUP by fin_data.MID,fin_data.Gender
HAVING Gender='Male' and count(Gender)>=1"""

print(grader_5ab(query_5ab))


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

```
          MID  Gender  Count
0  tt0021594    None      0
1  tt0021594  Female      3
2  tt0021594    Male      5
3  tt0026274    None      0
4  tt0026274  Female     11
5  tt0026274    Male      9
6  tt0027256    None      0
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: 1.99 s
```

In [22]:

```
%%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 M.year,Count(tab1.MID) as Count_Movie from (SELECT fin_data.MID fro
m (SELECT M.MID,person_data.Gender from (SELECT MC.MID,P.Gender from M_Cast MC
inner JOIN Person P
on ltrim(MC.PID)=P.PID) as person_data
inner join Movie M
on person_data.MID=M.MID) fin_data
GROUP by fin_data.MID,fin_data.Gender
EXCEPT
 SELECT fin_data.MID from (SELECT M.MID,person_data.Gender from (SELECT MC.MID,P.Gender
from M_Cast MC
inner JOIN Person P
on ltrim(MC.PID)=P.PID) as person_data
inner join Movie M
on person_data.MID=M.MID) fin_data
GROUP by fin_data.MID,fin_data.Gender
HAVING Gender='Male' and count(Gender)>=1) as tab1
INNER join
Movie M
on tab1.MID=M.MID
GROUP by M.year"""
grader_5a(query5a)
```

```
   year  Count_Movie
0  1939            1
1  1999            1
2  2000            1
3  2018            1
Wall time: 1.88 s
```

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 [23]:

```
cursor.execute('DROP TABLE IF EXISTS temp.temp_female_movie;')
cursor.execute("""Create TEMP TABLE temp_female_movie as
SELECT M.year,count(tab1.MID) female_count from
Movie M
INNER join
(SELECT fin_data.MID from (SELECT M.MID,person_data.Gender from (SELECT MC.MID,P.Gender
from M_Cast MC
inner JOIN Person P
on ltrim(MC.PID)=P.PID) as person_data
inner join Movie M
on person_data.MID=M.MID) fin_data
GROUP by fin_data.MID,fin_data.Gender
EXCEPT
 SELECT fin_data.MID from (SELECT M.MID,person_data.Gender from (SELECT MC.MID,P.Gender
from M_Cast MC
inner JOIN Person P
on ltrim(MC.PID)=P.PID) as person_data
inner join Movie M
on person_data.MID=M.MID) fin_data
GROUP by fin_data.MID,fin_data.Gender
HAVING Gender='Male' and count(Gender)>=1) as tab1
on tab1.MID=M.MID
GROUP by M.year
HAVING count(tab1.MID)>0;
""")
```

Out[23]:

```
<sqlite3.Cursor at 0x13aa3676dc0>
```

In [24]:

```
%%time
def grader_5b(q5b):
    q5b_results  = pd.read_sql_query(q5b,conn)
    print(q5b_results.head(10))
    assert (q5b_results.shape == (4,3))

query5b = """ SELECT ltrim(tfm.year) Year,cast(tfm.female_count as double)/cast(count(M
1.MID) as double) as Percentage_Female_Only_Movie, count(M1.MID) as Total_Movies from
Movie M1
left JOIN
temp_female_movie tfm
on ltrim(M1.year)=ltrim(tfm.year)
group by ltrim(M1.year)
having tfm.year is not NULL"""
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
Wall time: 24 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 [25]:

```
%%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,tab.count_PID as count from
Movie M
inner JOIN
(SELECT ltrim(mc.MID) MID,count(distinct mc.PID) count_PID from
M_Cast mc
GROUP by mc.MID) tab
on M.MID=tab.MID
order by tab.count_PID 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
Wall time: 345 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 [26]:

```
%%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 distinct M.Year,count(DISTINCT(M.MID)) as count FROM
Movie M
GROUP By ltrim(M.year)"""
grader_7a(query7a)



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

```
   year  count
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: 16 ms
```

In [27]:

```
%%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 = """
    SELECT DISTINCT tab1.year,tab1.count TotalMovies,tab2.year,tab2.count TotalMovies f
rom (SELECT ltrim(M.Year) Year,count(DISTINCT(M.MID)) as count FROM
Movie M
GROUP By ltrim(M.year))tab1
inner JOIN
(SELECT ltrim(M.Year) Year,count(DISTINCT(M.MID)) as count FROM
Movie M
GROUP By ltrim(M.year))tab2
on tab1.year<=tab2.year and tab2.year<=ltrim(tab1.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
```

```
   Year  TotalMovies  Year  TotalMovies
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: 42.5 ms
```

In [28]:

```
%%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 Decade,max(Movie_count) from (SELECT mq.year||"-"||max(mq.year2) Dec
ade,sum(TotalMovies2) Movie_count from (SELECT  tab1.year,tab1.count_mov TotalMovies1,t
ab2.year year2,tab2.count_mov TotalMovies2 from (SELECT ltrim(M.Year) Year,count(DISTIN
CT(M.MID)) as count_mov FROM
Movie M
GROUP By ltrim(M.year))tab1
inner JOIN
(SELECT ltrim(M.Year) Year,count(DISTINCT(M.MID)) as count_mov FROM
Movie M
GROUP By ltrim(M.year))tab2
on tab1.year<=tab2.year and tab2.year<=ltrim(tab1.year+9)
) as mq
GROUP by mq.year) ex"""
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  max(Movie_count)
0  2008-2017             1203
Wall time: 40 ms
```

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

In [29]:

```
%%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 DISTINCT mc.PID,md.PID,count(MC.MID) Movies from
M_Cast MC
inner JOIN
M_Director MD
on mc.MID=md.MID
group by mc.PID,md.PID"""
grader_8a(query8a)

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

```
          PID        PID  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: 1.41 s
```

In [30]:

```
%%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 = """select P.Name,tab.Movies from (SELECT Actor,Count_movies Movies from(SELECT
Actor,Director,Movies Count_movies,rank() OVER(PARTITION by Actor order by Movies desc)
movie_rank from (SELECT DISTINCT ltrim(mc.PID) Actor,ltrim(md.PID) Director,count(MC.MI
D) Movies from
M_Cast MC
inner JOIN
M_Director MD
on mc.MID=md.MID
group by mc.PID,md.PID) t1
) t2
where movie_rank=1 and Director='nm0007181') tab
inner join Person p
on tab.Actor=ltrim(p.PID)
order by tab.Movies DESC;"""
grader_8(query8)
```

```
                Name  Movies
0        Jagdish Raj      11
1   Manmohan Krishna      10
2           Iftekhar       9
3      Shashi Kapoor       7
4     Waheeda Rehman       5
5      Rakhee Gulzar       5
6     Achala Sachdev       4
7        Neetu Singh       4
8           Ravikant       4
9    Parikshat Sahni       3
(245, 2)
Wall time: 2.64 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 [31]:

```
s1_drop="DROP TABLE IF EXISTS temp.s1;"
s2_drop="DROP TABLE IF EXISTS temp.s2;"
s1="""create TEMP table s1 as
select ltrim(mc1.MID) MID,ltrim(mc1.PID) PID from M_Cast mc1
inner join (
SELECT ltrim(MID) MID from M_Cast mc
where ltrim(mc.PID)='nm0451321') tab
on ltrim(mc1.MID)=tab.MID"""
s2="""create temp table s2 as
SELECT DISTINCT mc.MID from s1
INNER join M_Cast mc
on s1.PID=ltrim(mc.PID)"""
cursor.execute(s1_drop)
cursor.execute(s2_drop)
cursor.execute(s1)
cursor.execute(s2)
```

Out[31]:

```
<sqlite3.Cursor at 0x13aa3676dc0>
```

In [33]:

```
%%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 = """ SELECT P.Name from Person P
inner join (
SELECT DISTINCT ltrim(mm.PID) PID from s2
INNER join M_Cast mm
on s2.MID=mm.MID
EXCEPT
SELECT PID from s1) ll
on ltrim(P.PID)=ll.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)
Wall time: 1.14 s
```

In [ ]: