SQL Assignment

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

from IPython.display import display, HTML
```

In [3]:
```python
conn = sqlite3.connect("/Users/pinakshome/Downloads/Db-IMDB-Assignment.db")
cur=conn.cursor()
```

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 [4]:
```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

	cid	name	type	notnull	dflt_value	pk
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

```
----------------------------------------------------------------------------------------------------
```

In [74]:
```
pd.read_sql_query('''SELECT Name FROM Person where trim(Name) Like 'Yash Chopra' ''' ,conn)
```

Out[74]:

	Name
0	Yash Chopra

In [5]:
```
pd.read_sql_query('SELECT COUNT(*) FROM Person',conn)
```

Out[5]:

	COUNT(*)
0	37566

Preprocessing year in movies

In [6]:
```
pd.read_sql_query('SELECT year FROM Movie',conn)
```

Out[6]:

	year
0	2018
1	2018

2	2018
3	2012
4	2018
...	...
3468	1986
3469	1993
3470	2006
3471	1939
3472	1994

3473 rows × 1 columns

we will use the cast substring when selecting year from now

We will use trim(whatever column name) to remove trailing spaces

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 [7]:
```
%%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,m.year FROM Person p JOIN M_Director md ON p.PID=md.PID join Movie m on m.MID=mc
         WHERE m.MID IN (SELECT m.MID FROM Movie m JOIN M_Genre mg on m.MID=mg.MID where ((SUBSTR(m.year,-4,4)%4=
         and SUBSTR(m.year,-4,4)%100!=0) or SUBSTR(m.year,-4,4)%400=0) AND mg.GID IN (SELECT g.GID FROM
         Genre g JOIN M_genre mg on g.GID=mg.GID WHERE g.Name LIKE '%Comedy%'))'''

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 65.8 ms, sys: 6.87 ms, total: 72.7 ms
Wall time: 74.7 ms
```

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

In [8]:
```
%%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 M_Cast m JOIN Person p on Trim(m.PID)=Trim(p.PID) where m.MID IN
             (SELECT m.MID FROM M_Cast m JOIN Movie mm on m.MID=mm.MID WHERE mm.title LIKE 'Anand')"""
grader_2(query2)
```

```
                Name
0      Rajesh Khanna
1   Amitabh Bachchan
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 161 ms, sys: 6.63 ms, total: 168 ms
Wall time: 168 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 [9]:
```
%%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 234 ms, sys: 11.4 ms, total: 246 ms
Wall time: 243 ms
```

```
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))

query3 = '''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

INTERSECT

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'''



grader_3(query3)
```

```
         PID
0  nm0000821
1  nm0003987
2  nm0004334
3  nm0004429
4  nm0004432
5  nm0004433
6  nm0004434
7  nm0004435
8  nm0004564
9  nm0004569
CPU times: user 196 ms, sys: 6.6 ms, total: 203 ms
Wall time: 202 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 p.PID, Count(*) FROM M_Director m Join Person p on (m.PID)=p.PID Join Movie
              mm on mm.MID=m.MID group by p.PID"""
print(grader_4a(query_4a))

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

```
         PID  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
```

In [30]:
```
%%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 p.Name,COUNT(m.PID) Movie_count from Person p
            join M_Director m on Trim(p.PID)=Trim(m.PID) JOIN
            Movie mm on mm.MID=m.MID GROUP BY m.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
CPU times: user 14.4 s, sys: 51 ms, total: 14.5 s
Wall time: 14.5 s
```

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

In [3]:
```
%%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 c.MID, p.Gender, COUNT(*) AS Count
                FROM Person p JOIN M_Cast c
                ON (p.PID) = Trim(c.PID)
                GROUP BY 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))

query_5ab ="""SELECT c.MID, p.Gender, COUNT(*) Count
                FROM M_Cast c INNER JOIN Person p
                ON (p.PID) = trim(c.PID) where p.Gender='Male' GROUP BY c.MID, p.Gender Having Count>=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      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 285 ms, sys: 21 ms, total: 306 ms
Wall time: 306 ms
```

In [94]:
```
%%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 SUBSTR(year,-4,4), COUNT(MID) FROM Movie where MID NOT IN
      (SELECT MID FROM (SELECT c.MID, p.Gender, COUNT(*) Count FROM
                          M_Cast c INNER JOIN Person p
                          ON (p.PID) = trim(c.PID) where p.Gender='Male' GROUP BY c.MID, p.Gender Having Count>=1
                          GROUP BY MID'''
grader_5a(query5a)
```

```
  SUBSTR(year,-4,4)  COUNT(MID)
0              1999           1
1              2000           1
2              1939           1
3              2018           1
CPU times: user 128 ms, sys: 4.64 ms, total: 133 ms
Wall time: 131 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 [145…
```
#source: https://stackoverflow.com/questions/57743348/sql-query-imdb-data-to-count-the-total-movies-with-only-fem

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 female_count.Year, ((female_count.OnlyF)*100)/total_count.TT,total_count.TT
FROM

((SELECT SUBSTR(year,-4,4) Year, COUNT(MID) OnlyF FROM Movie where MID NOT IN
      (SELECT MID FROM (SELECT c.MID, p.Gender, COUNT(*) Count FROM
                          M_Cast c INNER JOIN Person p
                          ON (p.PID) = trim(c.PID) where p.Gender='Male' GROUP BY c.MID, p.Gender Having Count>=1
                          GROUP BY MID) female_count,
 (SELECT Movie.year MM,count(*) TT FROM Movie group by Movie.year) total_count)
WHERE female_count.year=total_count.MM'''
grader_5b(query5b)
```

```
   Year  ((female_count.OnlyF)*100)/total_count.TT  TT
0  1999                                          1  66
1  2000                                          1  64
2  1939                                         50   2
3  2018                                          1  93
```

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 [156…
```
%%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(*) J from Movie m join M_Cast mc on m.MID=mc.MID Group by m.MID ORDER BY
             J desc"""
grader_6(query6)
```

```
                      title    J
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 129 ms, sys: 8.8 ms, total: 138 ms
Wall time: 136 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 distinct(SUBSTR(year,-4,4)), Count(SUBSTR(year,-4,4)) From Movie GROUP BY SUBSTR(year,-4,4)''
grader_7a(query7a)


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

```
  (SUBSTR(year,-4,4))  Count(SUBSTR(year,-4,4))
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 8.33 ms, sys: 1.39 ms, total: 9.72 ms
Wall time: 10 ms
```

In [42]:
```
%%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 m.Y, m.Z,n.X,n.C from
            (SELECT year Y, COUNT(*) Z From Movie group by year) m ,
            (SELECT year X, COUNT(*) C From Movie group by year) n WHERE
             n.X<=m.Y+9 and n.X>=m.Y"""
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
```

```
      Y  Z     X  C
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 14.3 ms, sys: 2.35 ms, total: 16.7 ms
Wall time: 15.5 ms
```

In [24]:
```
%%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 (y.year),Count(*) aa FROM
             (SELECT DISTINCT(year) FROM Movie) y
              JOIN Movie m on m.year>=y.year and m.year<=y.year+9
              group by y.year order by aa 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 2008-2017
```

```
   year    aa
0  2008  1126
CPU times: user 76.5 ms, sys: 3.74 ms, total: 80.3 ms
Wall time: 80.5 ms
```

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

In [44]:

```
%%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 mm.PID,m.PID,count(*) FROM M_Director mm
              JOIN M_Cast m on mm.MID=m.MID group by mm.PID,m.PID"""
grader_8a(query8a)

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

```
         PID         PID  count(*)
0  nm0000180   nm0000027         1
1  nm0000180   nm0001114         1
2  nm0000180   nm0001919         1
3  nm0000180   nm0006762         1
4  nm0000180   nm0030062         1
5  nm0000180   nm0038970         1
6  nm0000180   nm0051856         1
7  nm0000180   nm0085966         1
8  nm0000180   nm0097889         1
9  nm0000180   nm0125497         1
CPU times: user 326 ms, sys: 36.3 ms, total: 362 ms
Wall time: 376 ms
```

In [84]:

```
%%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 Actor_Name, mov_cnt Movie_count_with_yash_chopra
            FROM
                (SELECT actor act_id, dir_name, movies mov_cnt
                    FROM

                    (SELECT trim(mc.PID) actor, p.Name dir_name, trim(md.PID) director, COUNT(*) as movies
                        FROM M_Cast mc
                        JOIN M_Director md ON trim(mc.MID) = md.MID
                        JOIN Person p ON director = p.PID
                        GROUP BY actor,director
                    )

            WHERE (act_id,mov_cnt) IN

                    (SELECT actor act_id, MAX(movies)
                        FROM
                        (SELECT trim(mc.PID) actor, trim(md.PID) director, COUNT(*) as movies
                            FROM M_Cast mc
                            JOIN M_Director md ON trim(mc.MID) = md.MID
                            GROUP BY actor,director
                        )
                    GROUP BY act_id
                    )
            AND dir_name LIKE '%Yash Chopra%'
            )
```

```
        JOIN Person p ON act_id = p.PID
        ORDER BY Movie_count_with_yash_chopra DESC
            '''
grader_8(query8)
```

```
          Actor_Name  Movie_count_with_yash_chopra
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 496 ms, sys: 22.3 ms, total: 518 ms
Wall time: 520 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 [55]:
```
%%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 = """SELECT distinct(p.PID) FROM M_Cast m join Person p on trim(m.PID)=(p.PID) where m.MID IN
            (SELECT (m.MID) from M_Cast m join Person p on (p.PID)=trim(m.PID)
            where trim(p.Name) LIKE '%Shah Rukh Khan%') AND  trim(p.Name) != 'Shah Rukh Khan' """
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)
CPU times: user 199 ms, sys: 7.09 ms, total: 206 ms
Wall time: 209 ms
```

In [79]:
```
%%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_act AS

            (SELECT distinct(p.PID) a1 FROM M_Cast m join Person p on trim(m.PID)=(p.PID) where m.MID IN
            (SELECT (m.MID) from M_Cast m join Person p on (p.PID)=trim(m.PID)
             where trim(p.Name) LIKE '%Shah Rukh Khan%') AND  trim(p.Name) != 'Shah Rukh Khan'),

             S1_mov AS

             (SELECT distinct(m.MID) M1 from M_Cast m join Person p on (p.PID)=trim(m.PID)
             where trim(p.PID) IN  (SELECT distinct(p.PID) a1 FROM M_Cast m join Person p on trim(m.PID)=(p.PID)
            (SELECT (m.MID) from M_Cast m join Person p on (p.PID)=trim(m.PID)
              where trim(p.Name) LIKE '%Shah Rukh Khan%') AND  trim(p.Name) != 'Shah Rukh Khan')  and m.MID NOT IN
              where trim(p.Name) LIKE '%Shah Rukh Khan%'))


              SELECT S2.a2 from (SELECT distinct(p.PID) a2 FROM M_Cast m join Person p on trim(m.PID)=(p.PID)
              where m.MID IN S1_mov) S2 where S2.a2 Not in S1_act









              """
grader_9(query9)
```

```
          a2
0  nm2539953
1  nm0922035
2  nm0324658
3  nm0943079
4  nm0000218
5  nm0001394
6  nm0929654
7  nm3116102
8  nm3248891
9  nm2418809
(25698, 1)
CPU times: user 771 ms, sys: 8.94 ms, total: 780 ms
Wall time: 779 ms
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js