SQL Assignment

In [1]:

```
import pandas as pd
import sqlite3

from IPython.display import display, HTML
```

In [2]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [3]:

```
# Note that this is not the same db we have used in course videos, please downlo
ad from this link
# https://drive.google.com/file/d/1O-1-L1DdNxEK6O6nG2jS31MbrMh-OnXM/view?usp=sha
ring
```

In [2]:

```
conn = sqlite3.connect("/content/drive/MyDrive/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

Schema of new_records

	cid	name	type	notnull	dflt_value	pk
0	0	index	INT	0	None	0
1	1	MID	TEXT	0	None	0
2	2	PID	TEXT	0	None	0
3	3	ID	INT	0	None	0
4	4	index:1	INT	0	None	0
5	5	MID:1	TEXT	0	None	0
6	6	title	TEXT	0	None	0
7	7	year	TEXT	0	None	0
8	8	rating	REAL	0	None	0
9	9	num_votes	INT	0	None	0
10	10	index:2	INT	0	None	0
11	11	PID:1	TEXT	0	None	0
12	12	Name	TEXT	0	None	0
13	13	Gender	TEXT	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 [48]:

```
%time
def grader_1(q1):
    q1_results  = pd.read_sql_query(q1,conn)
    print(q1_results)
    assert (q1_results.shape == (232,3))

query1 = """ SELECT Movie.title, CAST(SUBSTR(Movie.year,-4) as integer) year1, P
erson.Name
            FROM Movie, M_Genre, Genre, M_Director, Person
            WHERE Movie.MID = M_Genre.MID
            AND M_Genre.GID = Genre.GID
            AND Genre.Name LIKE "%Comedy%"
            AND ((year1 % 4 = 0 AND year1 % 100 <> 0) OR (year1 % 400 = 0))
            AND Movie.MID = M_Director.MID
            AND M_Director.PID = Person.PID"""
grader_1(query1)
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 5.25 µs
                                 title  year1                    Nam
e
0                          Mastizaade   2016            Milap Zaver
i
1    Harold & Kumar Go to White Castle   2004            Danny Leine
r
2                  Gangs of Wasseypur   2012          Anurag Kashya
p
3          Around the World in 80 Days   2004            Frank Corac
i
4               The Accidental Husband   2008           Griffin Dunn
e
..                                 ...    ...
...
227                       Let's Enjoy   2004   Siddharth Anand Kuma
r
228                           Sathyam   2008         Amma Rajasekha
r
229                     Tandoori Love   2008           Oliver Paulu
s
230                       Le Halua Le   2012             Raja Chand
a
231                 Raja Aur Rangeeli   1996        K.S. Prakash Ra
o

[232 rows x 3 columns]
```

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

In [49]:

```
%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 distinct TRIM(PID, ' ') from M_Cast
                WHERE MID
                IN(
                  SELECT MID FROM Movie
                  WHERE title LIKE "Anand"
                )
            )"""
grader_2(query2)
```

```
CPU times: user 1e+03 ns, sys: 1 µs, total: 2 µs
Wall time: 4.77 µs
                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
```

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)<197
0)
) 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
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 7.63 µs
(4942, 1)
(62570, 1)
True
```

In [12]:

```
import pandas as pd
%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
    ACTORS_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 UNSIGNED) < 1970
    ),

    ACTORS_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 UNSIGNED) > 1990
    )

    SELECT DISTINCT TRIM(P.Name) Actor_Name
    FROM ACTORS_BEFORE_1970 A_1970  JOIN    ACTORS_AFTER_1990 A_1990
    ON      A_1970.PID = A_1990.PID JOIN    Person P
    ON      A_1970.PID = TRIM(P.PID)
"""

grader_3(query3)
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 6.2 µs
         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
```

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 [8]:

```
%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_Count
              FROM M_Director GROUP BY PID """
print(grader_4a(query_4a))

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

```
CPU times: user 3 µs, sys: 1 µs, total: 4 µs
Wall time: 6.68 µs
         PID  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
```

In [9]:

```
%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 DISTINCT TRIM(P.NAME) Director_Name, NM.Movie_Count
             FROM(
              SELECT PID, COUNT(MID) Movie_Count
              FROM M_Director GROUP BY PID
              HAVING Movie_Count >= 10
            ) NM JOIN    PERSON P
                 ON TRIM(NM.PID) = TRIM(P.PID)
                 ORDER BY NM.Movie_Count DESC """
grader_4(query4)
```

```
CPU times: user 1e+03 ns, sys: 1e+03 ns, total: 2 µs
Wall time: 4.53 µs
          Director_Name  Movie_Count
0          David Dhawan           39
1          Mahesh Bhatt           35
2          Priyadarshan           30
3       Ram Gopal Varma           30
4          Vikram Bhatt           29
5  Hrishikesh Mukherjee           27
6           Yash Chopra           21
7       Basu Chatterjee           19
8        Shakti Samanta           19
9          Subhash Ghai           18
```

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

In [28]:

```
%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  M_Cast.MID, Person.Gender, count() as Count FROM M_Cast, P
erson
              WHERE Person.PID = TRIM(M_Cast.PID)
              GROUP BY M_Cast.MID, Person.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  M_Cast.MID, Person.Gender, count() as Count FROM M_Cast, P
erson
              WHERE Person.PID = TRIM(M_Cast.PID)
              AND Person.Gender='Male'
              GROUP BY M_Cast.MID, Person.Gender
     """

print(grader_5ab(query_5ab))


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

```
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 7.63 µs
         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
```

In [11]:

```
%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 year, count(MID) as Female_Cast_Only_Movies FROM Movie
              WHERE MID NOT IN (
              SELECT  DISTINCT M_Cast.MID FROM M_Cast, Person
              WHERE Person.PID = TRIM(M_Cast.PID)
              AND Person.Gender IN ('Male', 'None' )
              )
              GROUP BY year
 """
grader_5a(query5a)
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 6.68 µs
     year  Female_Cast_Only_Movies
0    1939                        1
1    1999                        1
2    2000                        1
3  I 2018                        1
```

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 [10]:

```
%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
              Movie_Person_MCast
              AS (
              SELECT SUBSTR(Movie.year, -4) as year, Movie.MID, Person.Gender FR
OM M_Cast, Movie, Person
              WHERE Movie.MID = M_Cast.MID
              AND Person.PID = TRIM(M_Cast.PID)
              ),

              AllGenderCount
              AS(
              SELECT year, MID, Gender, CASE WHEN Gender='Female' THEN '1' WHEN
 Gender='Male' THEN '0' ELSE '0' END AS Result_Gender
              FROM Movie_Person_MCast
              ),

              Intermediate_Counts as
              (
              Select year,
              Sum(Result_Gender) as Female_Count,
              Count(*) as Total_Count
              From AllGenderCount
              GROUP BY year, MID
              ),

              relevant_movies_counts as (
                select year, count(*) as total_movies, sum(case when Female_Coun
t=Total_Count Then '1' else '0' end) as female_movies
                from Intermediate_Counts group by year
              )

              select year, female_movies*100.0/total_movies as Percentage_Female
_Only_Movie, total_movies
              from relevant_movies_counts
              where Percentage_Female_Only_Movie > 0"""
grader_5b(query5b)
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 5.25 µs
   year  Percentage_Female_Only_Movie  total_movies
0  1939                     50.000000             2
1  1999                      1.515152            66
2  2000                      1.562500            64
3  2018                      0.961538           104
```

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 [42]:

```
%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
              Movie_Person_MCast
              AS (
              SELECT Movie.MID, Movie.title, Person.PID FROM M_Cast, Movie, Pers
on
              WHERE Movie.MID = M_Cast.MID
              AND Person.PID = TRIM(M_Cast.PID)
              )

          SELECT title, count(distinct PID) as count  FROM Movie_Person_MCast Gr
oup by MID  ORDER BY count DESC  """
grader_6(query6)
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 5.72 µs
                        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
```

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 [65]:

```
%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 SUBSTR(Movie.year, -4) as year1, count(MID) FROM Movie  GRO
UP BY year1"""
grader_7a(query7a)

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

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 5.48 µs
  year1  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
```

In [111]:

```
%time
def grader_7b(q7b):
    q7b_results  = pd.read_sql_query(q7b,conn)
    print(q7b_results.head(10))
    assert (q7b_results.shape == (713, 4))

query7b = """
          WITH year_wise_count AS(
          SELECT cast(SUBSTR(Movie.year, -4) as integer) as year1, count(MID) as
MID_count FROM Movie  GROUP BY year1)

          SELECT y1.year1 as decade_start, y2.year1 as decade_years,y1.MID_Count
as _MID_count, y2.MID_count as MID_Count FROM year_wise_count y1, year_wise_coun
t y2
          WHERE y2.year1 <= y1.year1+9  AND  y2.year1 >= y1.year1
          """
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
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 8.11 µs
   decade_start  decade_years  _MID_count  MID_Count
0          1931          1931           1          1
1          1931          1936           1          3
2          1931          1939           1          2
3          1936          1936           3          3
4          1936          1939           3          2
5          1936          1941           3          1
6          1936          1943           3          1
7          1939          1939           2          2
8          1939          1941           2          1
9          1939          1943           2          1
```

In [120]:

```
%time
def grader_7(q7):
    q7_results  = pd.read_sql_query(q7,conn)
    print(q7_results.head(10))
    assert (q7_results.shape == (1, 2))

query7 = """ WITH year_wise_count AS(
          SELECT cast(SUBSTR(Movie.year, -4) as integer) as year1, count(MID) as
MID_count FROM Movie  GROUP BY year1),

          decade_table as(
          SELECT y1.year1 as decade_start, y1.year1+9 as decade_end, y2.year1 as
decade_years,y1.MID_Count as _MID_count, y2.MID_count as MID_Count FROM year_wis
e_count y1, year_wise_count y2
          WHERE y2.year1 <= y1.year1+9  AND  y2.year1 >= y1.year1
          )

          SELECT CAST(decade_start as text)||'-'||CAST(decade_end as text) as de
cade, sum(MID_Count) as MID_Count from decade_table group by decade_start
          ORDER BY MID_Count 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
```

```
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 8.11 µs
      decade  MID_Count
0  2008-2017       1203
```

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

In [11]:

```
%time
def grader_8a(q8a):
    q8a_results  = pd.read_sql_query(q8a,conn)
    print(q8a_results.head(10))
    assert (q8a_results.shape == (73408, 3))

# Write a query that will results in number of movies actor-director worked toge
ther
query8a = """ SELECT M_Cast.PID as actor, M_Director.PID as director, count(Movi
e.MID) as movies
              FROM M_Cast, Movie, M_Director
              WHERE Movie.MID = M_Cast.MID
              AND Movie.MID = M_Director.MID
              GROUP BY actor, director
          """

q8a_results  = pd.read_sql_query(query8a,conn)
grader_8a(query8a)

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

```
CPU times: user 3 µs, sys: 1 µs, total: 4 µs
Wall time: 8.11 µs
        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
```

In [44]:

```
%time

def grader_8(q8):
    q8_results  = pd.read_sql_query(q8,conn)
    print(q8_results.head(10))
    print(q8_results.shape)
    assert (q8_results.shape == (245, 2))

query8 = """
          WITH yash_chopra_pid as (
            SELECT distinct M_Director.PID from M_Director, Person
            WHERE Person.Name LIKE '%yash chopra%'
            AND Person.PID= M_Director.PID
          ),

          actor_directo_movies as(
            SELECT M_Cast.PID as actor, M_Director.PID as director, count(distin
ct M_Cast.MID) as movies
            FROM M_Cast, M_Director
            WHERE M_Director.MID = M_Cast.MID
            AND M_Director.PID NOT IN (SELECT PID FROM yash_chopra_pid)
            GROUP BY actor, director
          ),

          actor_max as (
            SELECT actor, max(movies) as max_count
            from actor_directo_movies
            group by actor
          ),

          actor_directo_movies_yash as(
            SELECT M_Cast.PID as actor, M_Director.PID as director, count(distin
ct M_Cast.MID) as movies
            FROM M_Cast, M_Director
            WHERE M_Director.MID = M_Cast.MID
            AND M_Director.PID IN (SELECT PID FROM yash_chopra_pid)
            GROUP BY actor, director
          ),

          relevant_actors as (
            SELECT actor_directo_movies_yash.actor, actor_directo_movies_yash.mo
vies as yash_count
            from actor_directo_movies_yash LEFT OUTER JOIN actor_max
            ON actor_directo_movies_yash.actor = actor_max.actor
            WHERE actor_max.max_count IS NULL OR actor_directo_movies_yash.movie
s >= actor_max.max_count
          )

          SELECT Person.Name, relevant_actors.yash_count FROM relevant_actors, P
erson
          WHERE TRIM(relevant_actors.actor) = Person.PID
          ORDER BY relevant_actors.yash_count DESC


          """
grader_8(query8)
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 6.91 µs
                Name  yash_count
0        Jagdish Raj          11
1   Manmohan Krishna          10
2           Iftekhar           9
3      Shashi Kapoor           7
4      Rakhee Gulzar           5
5     Waheeda Rehman           5
6           Ravikant           4
7     Achala Sachdev           4
8        Neetu Singh           4
9      Leela Chitnis           3
(245, 2)
```

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 [16]:

```
%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
            SHAHRUKH_0 AS                            --[PID] where Person.Name co
ntains Shahrukh
            (
                SELECT P.PID PID
                FROM Person P
                WHERE P.Name like '%Shah Rukh Khan%'
            ),

            SHAHRUKH_1_MOVIES AS                    -- [MID, PID] where Shahrukh
 PIDs worked
            (
            SELECT DISTINCT MC.MID MID, S0.PID
            FROM M_Cast MC, SHAHRUKH_0 S0
            WHERE TRIM(MC.PID) = S0.PID
            ),

            SHAHRUKH_1_ACTORS AS                    --[PID] of actors worked in s
ame Movies MID as Shahrukh, except Shahrukh
            (
            SELECT DISTINCT TRIM(MC.PID) PID
            FROM M_Cast MC, SHAHRUKH_1_MOVIES S1M
            WHERE TRIM(MC.MID) = S1M.MID
            AND TRIM(MC.PID) <> S1M.PID
            )

            SELECT * FROM SHAHRUKH_1_ACTORS  """
grader_9a(query9a)
# using the above query, you can write the answer to the given question

# selecting actors who acted with srk (S1)
# selecting all movies where S1 actors acted, this forms S2 movies list
# selecting all actors who acted in S2 movies, this gives us S2 actors along wit
h S1 actors
# removing S1 actors from the combined list of S1 & S2 actors, so that we get on
ly S2 actors
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 5.25 µs
         PID
0  nm0004418
1  nm1995953
2  nm2778261
3  nm0631373
4  nm0241935
5  nm0792116
6  nm1300111
7  nm0196375
8  nm1464837
9  nm2868019
(2382, 1)
```

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 = """ WITH
            SHAHRUKH_0 AS                              --[PID] where Person.Name co
ntains Shahrukh
            (
                SELECT P.PID PID
                FROM Person P
                WHERE P.Name LIKE '%Shah Rukh Khan%'
            ),

            SHAHRUKH_1_MOVIES AS                    -- [MID, PID] where Shahrukh
 PIDs worked
            (
            SELECT DISTINCT TRIM(MC.MID) MID, S0.PID
            FROM M_Cast MC, SHAHRUKH_0 S0
            WHERE TRIM(MC.PID) = S0.PID
            ),

            SHAHRUKH_1_ACTORS AS                    --[PID] of actors worked in s
ame Movies MID as Shahrukh, except Shahrukh
            (
            SELECT DISTINCT TRIM(MC.PID) PID
            FROM M_Cast MC, SHAHRUKH_1_MOVIES S1M
            WHERE TRIM(MC.MID) = S1M.MID
            AND TRIM(MC.PID) <> S1M.PID
            ),

            SHAHRUKH_2_MOVIES AS                     --[MID, PID] who worked with
actors worked with Shahrukh
            (
            SELECT DISTINCT TRIM(MC.MID) MID
            FROM M_Cast MC, SHAHRUKH_1_ACTORS S1A
            WHERE TRIM(MC.PID) = S1A.PID
            ),
            SHAHRUKH_2_ACTORS AS
            (
              SELECT DISTINCT TRIM(MC.PID) PID
              FROM M_Cast MC, SHAHRUKH_2_MOVIES S2M
              WHERE TRIM(MC.MID) = S2M.MID
            )


            SELECT P.Name ACTOR_NAME
            FROM Person P, SHAHRUKH_2_ACTORS S2A
            WHERE S2A.PID = P.PID
            AND P.PID NOT IN (SELECT PID FROM SHAHRUKH_1_ACTORS)
            AND P.PID NOT IN (SELECT PID FROM SHAHRUKH_0)
            """
grader_9(query9)
```

```
CPU times: user 2 µs, sys: 1e+03 ns, total: 3 µs
Wall time: 5.48 µs
               ACTOR_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)
```

1. Find Actors that were never unemployed for more than 3 years.

- Assumtions :

  A) I'm considering only people who have worked for more than one year.

  B) Considering the time period between min and max years of the actor. i.e if the actor has been working from 1990 to 2000 and the actor acted only in 1990, 1991 1996, 1998 and 2000. Then he is considered as unemployed for more than 3 yrs (> 3 yrs => atleast 4 years).

- Logic for solving : Calculate the total num of movies that the actor acted from his min year i.e 1990 to 1991, let say it's 3 and then calculate the total num of movies he acted from his min year 1990 to 1991+4 (1995) and it comes back as 3 since he hasn't made any movies between 1991 and 1995 , this means that he has been unemployed for more than 3 years (4 years) 1992,1993,1994,1995 , therefore we don't consider him.

In [28]:

```
%time
query_10 = """
          WITH NUM_OF_MOV_FOR_AN_ACTR_BY_YR AS
          (
          SELECT TRIM(MC.PID) PID, CAST(SUBSTR(year,-4) AS UNSIGNED) YEAR, COUNT
(DISTINCT TRIM(M.MID)) NUM_OF_MOV
          FROM M_Cast MC, Movie M
          WHERE MC.MID = M.MID
          GROUP BY TRIM(MC.PID), CAST(SUBSTR(year,-4) AS UNSIGNED)
          ORDER BY NUM_OF_MOV DESC
          ),

          ACTRS_FOR_MORE_THAN_ONE_YR AS
          (
          SELECT PID, COUNT(YEAR) AS NUM_OF_YEARS, MIN(YEAR) AS MIN_YEAR, MAX(YE
AR) AS MAX_YEAR
          FROM NUM_OF_MOV_FOR_AN_ACTR_BY_YR
          GROUP BY PID
          HAVING COUNT(YEAR) > 1
          ),

          NUM_OF_FOR_ACTR_W_MRE_THN_1_YR AS
          (
          SELECT NM.PID, NM.YEAR, NM.YEAR+4 AS YEAR_PLUS_4, NM.NUM_OF_MOV, AY.MI
N_YEAR, AY.MAX_YEAR
          FROM NUM_OF_MOV_FOR_AN_ACTR_BY_YR NM, ACTRS_FOR_MORE_THAN_ONE_YR AY
          WHERE NM.PID = AY.PID
          ),

          NUM_OF_MOV_TILL_DATE_BY_ACTOR AS
          (
          SELECT NA.PID, NY.YEAR, SUM(NA.NUM_OF_MOV) AS NUM_OF_MOV_TILL_THAT_YEA
R
          FROM NUM_OF_FOR_ACTR_W_MRE_THN_1_YR NA, NUM_OF_FOR_ACTR_W_MRE_THN_1_YR
NY
          WHERE NA.PID = NY.PID AND NA.YEAR BETWEEN NY.MIN_YEAR AND  NY.YEAR
          GROUP BY NA.PID, NY.YEAR
          ),

          NUM_OF_MV_BY_ACTR_BY_YR_PLS_4 AS
          (
          SELECT NA.PID,NY.YEAR, SUM(NA.NUM_OF_MOV) AS NUM_OF_MOV_TILL_AS_OF_YR_
PLS_4
          FROM NUM_OF_FOR_ACTR_W_MRE_THN_1_YR NA, NUM_OF_FOR_ACTR_W_MRE_THN_1_YR
NY
          WHERE NA.PID = NY.PID
          AND NA.YEAR BETWEEN NY.MIN_YEAR
          AND  NY.YEAR_PLUS_4
          AND NY.YEAR_PLUS_4 <= NY.MAX_YEAR
          GROUP BY NA.PID, NY.YEAR
          )

          SELECT DISTINCT TRIM(P.Name) AS ACTORS_NEVER_UNEMPLOYED_FOR_MORE_THAN_
3_YRS
          FROM Person P
          WHERE TRIM(P.PID) NOT IN
          (
          SELECT DISTINCT NMT.PID
          FROM NUM_OF_MOV_TILL_DATE_BY_ACTOR NMT, NUM_OF_MV_BY_ACTR_BY_YR_PLS_4
```

```
 NMP
          WHERE NMT.PID = NMP.PID
          AND NMT.YEAR = NMP.YEAR
          AND NMT.NUM_OF_MOV_TILL_THAT_YEAR = NMP.NUM_OF_MOV_TILL_AS_OF_YR_PLS_4
          )

          """

query_10 = pd.read_sql_query(query_10,conn)
print(query_10)
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 5.25 µs
      ACTORS_NEVER_UNEMPLOYED_FOR_MORE_THAN_3_YRS
0                                  Christian Bale
1                                  Cate Blanchett
2                              Benedict Cumberbatch
3                                     Naomie Harris
4                                       Andy Serkis
...                                             ...
32580                              Deepak Ramteke
32581                                Kamika Verma
32582                           Dhorairaj Bhagavan
32583                                 Nasir Shaikh
32584                                 Adrian Fulle

[32585 rows x 1 columns]
```