SQL Assignment

In [ ]:

```
import pandas as pd
import sqlite3

from IPython.display import display, HTML
```

In [ ]:

```
# 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/1O-1-L1DdNxEK6O6nG2jS31MbrMh-OnXM/view?usp=sharing
```

In [ ]:

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/dri
ve", force_remount=True).
```

In [ ]:

```
conn = sqlite3.connect("/content/drive/MyDrive/Db-IMDB-Assignment.db")
```

Overview of all tables

In [ ]:

```
tables = pd.read_sql_query("SELECT NAME AS 'Table_Name' FROM sqlite_master WHERE type='table' and name
not LIKE 'sqlite_%'",conn)
tables = tables["Table_Name"].values.tolist()
```

In [ ]:

```
tables
```

Out[ ]:

```
['Movie',
 'Genre',
 'Language',
 'Country',
 'Location',
 'M_Location',
 'M_Country',
 'M_Language',
 'M_Genre',
 'Person',
 'M_Producer',
 'M_Director',
 'M_Cast']
```

In [ ]:

```
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("-"*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

	cid	name	type	notnull	dflt_value	pk
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

	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

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 [ ]:

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

query1 = """SELECT PERSON.NAME, MOVIE.TITLE, MOVIE.YEAR
            FROM MOVIE
            JOIN M_DIRECTOR ON TRIM(M_DIRECTOR.MID) = TRIM(MOVIE.MID)
            JOIN PERSON ON TRIM(PERSON.PID) = TRIM(M_DIRECTOR.PID)
            WHERE TRIM(MOVIE.MID) IN (SELECT TRIM(M_GENRE.MID) FROM M_GENRE WHERE TRIM(M_GENRE.GID) IN
(SELECT TRIM(GENRE.GID) FROM GENRE WHERE TRIM(GENRE.NAME) LIKE '%Comedy%'))
            AND (((CAST(SUBSTR(TRIM(MOVIE.YEAR),-4) AS INTEGER)%4==0)
            AND (CAST(SUBSTR(TRIM(MOVIE.YEAR),-4) AS INTEGER)%100!=0))
            OR (CAST(SUBSTR(TRIM(MOVIE.YEAR),-4) AS INTEGER)%400==0))
        """
grader_1(query1)
```

```
CPU times: user 1.76 s, sys: 3.57 ms, total: 1.77 s
Wall time: 1.79 s
```

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

In [ ]:

```
%%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 PERSON.NAME ACTOR_NAMES
             FROM PERSON
             WHERE TRIM(PERSON.PID) IN (SELECT trim(M_CAST.PID) FROM M_CAST WHERE M_CAST.MID = (SELECT
MOVIE.MID FROM MOVIE WHERE TRIM(MOVIE.TITLE) LIKE 'Anand'))
        """
grader_2(query2)
```

```
        ACTOR_NAMES
0   Amitabh Bachchan
1      Rajesh Khanna
2      Sumita Sanyal
3         Ramesh Deo
4          Seema Deo
5     Asit Kumar Sen
6         Dev Kishan
7       Atam Prakash
8      Lalita Kumari
9             Savita
CPU times: user 21.8 ms, sys: 4.26 ms, total: 26.1 ms
Wall time: 32.3 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 [ ]:

```
%%time

def grader_3a(query_less_1970, query_more_1990):
    q3_a = pd.read_sql_query(query_less_1970,conn)
    q3_b = pd.read_sql_query(query_more_1990,conn)
    return (q3_a.shape == (4942,1)) and (q3_b.shape == (62570,1))

query_less_1970 =""" SELECT M_CAST.ID
                      FROM M_CAST
                      WHERE M_CAST.ID IN
                      (SELECT trim(M_CAST.ID) FROM M_CAST WHERE trim(M_CAST.MID) IN
                      (SELECT trim(MOVIE.MID) FROM MOVIE WHERE CAST(SUBSTR(TRIM(MOVIE.YEAR),-4) AS INTE
GER) < 1970 )) """

query_more_1990 =""" SELECT  M_CAST.ID
                     FROM M_CAST
                     WHERE M_CAST.ID IN
                     (SELECT trim(M_CAST.ID) FROM M_CAST WHERE trim(M_CAST.MID) IN
                     (SELECT trim(MOVIE.MID) FROM MOVIE WHERE CAST(SUBSTR(TRIM(MOVIE.YEAR),-4) AS INTEG
ER) > 1990 ))"""
print(grader_3a(query_less_1970, query_more_1990))

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

```
True
CPU times: user 170 ms, sys: 13.8 ms, total: 183 ms
Wall time: 186 ms
```

In [ ]:

```
%%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 PERSON.NAME ACTOR_NAME
             FROM PERSON
             WHERE TRIM(PERSON.PID) IN
             (SELECT TRIM(PID) AS ID FROM M_CAST WHERE ID IN
             (SELECT trim(M_CAST.ID) FROM M_CAST WHERE trim(M_CAST.MID) IN
             (SELECT trim(MOVIE.MID) FROM MOVIE WHERE CAST(SUBSTR(TRIM(MOVIE.YEAR),-4) AS INTEGER) > 19
90 ))

            INTERSECT
            SELECT TRIM(PID) AS ID
            FROM M_CAST
            WHERE ID IN
            (SELECT trim(M_CAST.ID) FROM M_CAST WHERE trim(M_CAST.MID) IN
            (SELECT trim(MOVIE.MID) FROM MOVIE WHERE CAST(SUBSTR(TRIM(MOVIE.YEAR),-4) AS INTEGER) < 197
0 )))
             """
grader_3(query3)
```

```
          ACTOR_NAME
0       Rishi Kapoor
1   Amitabh Bachchan
2             Asrani
3       Zohra Sehgal
4    Parikshat Sahni
5      Rakesh Sharma
6         Sanjay Dutt
7           Ric Young
8               Yusuf
9     Suhasini Mulay
CPU times: user 197 ms, sys: 4 ms, total: 201 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 [ ]:

```
%%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 AS DIRECTOR_ID ,COUNT(MID) AS 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
```

```
  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
CPU times: user 13.1 ms, sys: 94 µs, total: 13.2 ms
Wall time: 14.4 ms
```

In [ ]:

```
%%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 PERSON.NAME AS DIRECTOR_NAME , COUNT(M_DIRECTOR.MID) AS MOVIE_COUNT
             FROM M_DIRECTOR
             JOIN PERSON ON PERSON.PID = M_DIRECTOR.PID
             GROUP BY  M_DIRECTOR.PID HAVING MOVIE_COUNT > 9
             ORDER BY MOVIE_COUNT DESC """
grader_4(query4)
```

```
           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
CPU times: user 61.4 ms, sys: 870 µs, total: 62.3 ms
Wall time: 63.3 ms
```

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

In [ ]:

```
%%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
               JOIN PERSON ON 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
               JOIN PERSON ON PERSON.PID = TRIM(M_CAST.PID)
               GROUP BY M_CAST.MID ,PERSON.GENDER
               HAVING GENDER LIKE 'MALE' AND COUNT >0
           """

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 368 ms, sys: 5.88 ms, total: 374 ms
Wall time: 378 ms
```

In [ ]:

```
%%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(TRIM(MOVIE.YEAR),-4) AS YEAR,COUNT(*) FEMALE_CAST_ONLY
              FROM MOVIE
              WHERE TRIM(MOVIE.MID) NOT IN (SELECT DISTINCT(TRIM(M_CAST.MID)) AS MID_HAVING_MALE_OTHER
FROM M_CAST
              JOIN PERSON ON PERSON.PID = TRIM(M_CAST.PID)
              WHERE PERSON.GENDER LIKE 'MALE' OR PERSON.GENDER IS NULL)
              GROUP BY MOVIE.YEAR
```

```
            GROUP BY MOVIE.YEAR
            """
grader_5a(query5a)
```

```
   YEAR  FEMALE_CAST_ONLY
0  1939                 1
1  1999                 1
2  2000                 1
3  2018                 1
CPU times: user 159 ms, sys: 3.94 ms, total: 163 ms
Wall time: 165 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 [ ]:

```
%%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 T1.YEAR,CAST(T1.FEMALE_CAST_ONLY AS FLOAT)/CAST(T2.ALL_MOVIES AS FLOAT) Percentage
_Female_Only_Movie ,T2.ALL_MOVIES TOTAL_MOVIES
              FROM
              (SELECT SUBSTR(TRIM(MOVIE.YEAR),-4) AS YEAR,COUNT(*) FEMALE_CAST_ONLY
               FROM MOVIE
               WHERE TRIM(MOVIE.MID) NOT IN (SELECT DISTINCT(TRIM(M_CAST.MID)) AS MID_HAVING_MALE_OTHER
FROM M_CAST
               JOIN PERSON ON PERSON.PID = TRIM(M_CAST.PID)
               WHERE PERSON.GENDER LIKE 'MALE' OR PERSON.GENDER IS NULL)
               GROUP BY MOVIE.YEAR,MOVIE.MID ) T1

              JOIN

             (SELECT SUBSTR(TRIM(MOVIE.YEAR),-4) AS YEAR,COUNT(*) ALL_MOVIES
              FROM MOVIE
              GROUP BY CAST(SUBSTR(TRIM(MOVIE.YEAR),-4) AS INTEGER) ) T2
              ON T1.YEAR = T2.YEAR
            """
grader_5b(query5b)
```

```
   YEAR  Percentage_Female_Only_Movie  TOTAL_MOVIES
0  1939                      0.500000             2
1  1999                      0.015152            66
2  2000                      0.015625            64
3  2018                      0.009615           104
CPU times: user 163 ms, sys: 1.86 ms, total: 165 ms
Wall time: 167 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 [ ]:

```
%%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 MOVIE.TITLE,COUNT(DISTINCT(M_CAST.PID)) AS COUNT
```

```
query6 = """ SELECT MOVIE.TITLE,COUNT(DISTINCT(M_CAST.PID)) AS COUNT
             FROM M_CAST
             JOIN MOVIE ON MOVIE.MID = M_CAST.MID
             GROUP BY M_CAST.MID ORDER BY COUNT 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
CPU times: user 123 ms, sys: 8.98 ms, total: 132 ms
Wall time: 134 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 [ ]:

```
%%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 SUBSTR(TRIM(MOVIE.YEAR),-4) AS MOVIE_YEAR,COUNT(MOVIE.MID) AS MOVIE_COUNT
              FROM MOVIE
              GROUP BY SUBSTR(TRIM(MOVIE.YEAR),-4) """
grader_7a(query7a)

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

```
  MOVIE_YEAR  MOVIE_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
CPU times: user 10.3 ms, sys: 951 µs, total: 11.3 ms
Wall time: 11.7 ms
```

In [ ]:

```
%%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 T1.YEAR AS MOVIE_YEAR,T1.COUNT AS TOTAL_MOVIES,T2.YEAR AS MOVIE_YEAR,T2.COUNT AS
TOTAL_MOVIES
                FROM (SELECT CAST(SUBSTR(TRIM(MOVIE.YEAR),-4) AS INTEGER) AS  YEAR, COUNT(MOVIE.MID) AS
COUNT
                      FROM MOVIE
                      GROUP BY SUBSTR(TRIM(MOVIE.YEAR),-4) ) T1
                JOIN
                (SELECT  CAST(SUBSTR(TRIM(MOVIE.YEAR),-4) AS INTEGER) AS YEAR, COUNT(MOVIE.MID) AS COUN
T
                 FROM MOVIE
                 GROUP BY SUBSTR(TRIM(MOVIE.YEAR),-4) ) T2
                ON T2.YEAR <= T1.YEAR +9 AND T2.YEAR >=T1.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
```

```
   MOVIE_YEAR  TOTAL_MOVIES  MOVIE_YEAR  TOTAL_MOVIES
0        1931             1        1931             1
1        1931             1        1936             3
2        1931             1        1939             2
3        1936             3        1936             3
4        1936             3        1939             2
5        1936             3        1941             1
6        1936             3        1943             1
7        1939             2        1939             2
8        1939             2        1941             1
9        1939             2        1943             1
CPU times: user 16.7 ms, sys: 0 ns, total: 16.7 ms
Wall time: 17.3 ms
```

reference: https://stackoverflow.com/a/10538604/14259667

In [ ]:

```
%%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 FINAL.T1_YEAR AS DECADE,MAX(COUNT) AS MOVIES_COUNT
             FROM (SELECT T1.YEAR AS T1_YEAR,SUM(T2.COUNT) AS COUNT
                   FROM (SELECT CAST(SUBSTR(TRIM(MOVIE.YEAR),-4) AS INTEGER) AS  YEAR, COUNT(MOVIE.MID)
AS COUNT
                         FROM MOVIE
                         GROUP BY SUBSTR(TRIM(MOVIE.YEAR),-4)
                   ) T1
             JOIN
            (SELECT CAST(SUBSTR(TRIM(MOVIE.YEAR),-4) AS INTEGER) AS YEAR,COUNT(MOVIE.MID) AS COUNT
             FROM MOVIE
             GROUP BY SUBSTR(TRIM(MOVIE.YEAR),-4) ) T2
             ON T2.YEAR <= T1.YEAR +9 AND T2.YEAR >=T1.YEAR
             GROUP BY T1.YEAR ) FINAL"""

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
```

```
   DECADE  MOVIES_COUNT
0    2008          1203
CPU times: user 13.4 ms, sys: 0 ns, total: 13.4 ms
Wall time: 15.3 ms
```

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

In [ ]:

```
%%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_CAST.PID AS ACTORID, M_DIRECTOR.PID AS DIRECTORID, COUNT(DISTINCT(M_DIRECTOR.MID
)) AS MOVIES
              FROM M_DIRECTOR
              LEFT JOIN M_CAST ON M_CAST.MID = M_DIRECTOR.MID
              GROUP BY ACTORID,DIRECTORID
          """
grader_8a(query8a)

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

```
      ACTORID DIRECTORID  MOVIES
0   nm0000002  nm0496746       1
1   nm0000027  nm0000180       1
2   nm0000039  nm0896533       1
3   nm0000042  nm0896533       1
4   nm0000047  nm0004292       1
5   nm0000073  nm0485943       1
6   nm0000076  nm0000229       1
7   nm0000092  nm0178997       1
8   nm0000093  nm0000269       1
9   nm0000096  nm0113819       1
CPU times: user 460 ms, sys: 25 ms, total: 485 ms
Wall time: 487 ms
```

In [ ]:

```
%%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 PERSON.NAME,T2.MOVIE_COUNT
            FROM PERSON
            JOIN (SELECT T1.ACTORID AS ACTORID, T1.MOVIE_COUNT AS MOVIE_COUNT FROM
                   (SELECT M_CAST.PID AS ACTORID, M_DIRECTOR.PID AS DIRECTORID,COUNT(M_DIRECTOR.MID) AS
MOVIE_COUNT
                    FROM M_DIRECTOR
                    LEFT JOIN M_CAST ON M_CAST.MID = M_DIRECTOR.MID
                    GROUP BY ACTORID,DIRECTORID) T1
                  WHERE (T1.ACTORID,T1.MOVIE_COUNT) IN
                        (SELECT T.ACTORID ,MAX(T.MOVIE_COUNT) COUNT
                         FROM (SELECT M_CAST.PID AS ACTORID,M_DIRECTOR.PID AS DIRECTORID,COUNT(M_DIRECT
OR.MID) AS MOVIE_COUNT
                               FROM M_DIRECTOR
                               LEFT JOIN M_CAST ON M_CAST.MID = M_DIRECTOR.MID
                               GROUP BY ACTORID,DIRECTORID) T
                         GROUP BY T.ACTORID )
                  AND DIRECTORID = (SELECT TRIM(M_DIRECTOR.PID) FROM M_DIRECTOR WHERE TRIM(M_DIRECTOR.P
ID) IN
                                                (SELECT PID FROM PERSON WHERE TRIM(PERSON.NAME) LIKE '%Y
ASH%CHOPRA%') )
                  ORDER BY MOVIE_COUNT DESC
                  ) T2
            ON PERSON.PID = TRIM(T2.ACTORID)

         """
grader_8(query8)
```

```
               Name  MOVIE_COUNT
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)
CPU times: user 11.5 s, sys: 43.1 ms, total: 11.6 s
Wall time: 11.6 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 [ ]:

```
%%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(M_CAST.PID)  AS S1_PID
             FROM M_CAST
             WHERE TRIM(M_CAST.MID) IN
                (SELECT TRIM(MID) FROM M_CAST WHERE TRIM(PID) = (SELECT TRIM(PID) FROM PERSON WHERE PER
SON.NAME LIKE '%Shah Rukh Khan%'))
             AND TRIM(M_CAST.PID) != (SELECT TRIM(PID) FROM PERSON WHERE PERSON.NAME LIKE '%Shah Rukh K
han%') """
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
```

```
      S1_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 39.4 ms, sys: 1.01 ms, total: 40.4 ms
Wall time: 41.9 ms
```

In [ ]:

```
%%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 PERSON.NAME AS ACTOR_NAME
             FROM PERSON
             WHERE TRIM(PERSON.PID) IN (SELECT DISTINCT(TRIM(M_CAST.PID))
```

```
                                        FROM M_CAST
                                        WHERE TRIM(M_CAST.MID) IN
                                        (SELECT TRIM(MID) FROM M_CAST WHERE TRIM(PID) IN
                                        (SELECT DISTINCT(TRIM(M_CAST.PID)) FROM M_CAST WHERE TRIM(M_CA
ST.MID) IN
                                        (SELECT TRIM(MID) FROM M_CAST WHERE TRIM(PID) = (SELECT TRIM(P
ID) FROM PERSON WHERE PERSON.NAME LIKE '%Shah Rukh Khan%'))

                AND TRIM(M_CAST.PID) !=(SELECT TRIM(PID) FROM PERSON WHERE PERSON.NAME LIKE '%Shah Rukh
Khan%')))
                AND TRIM(M_CAST.PID) NOT IN (SELECT DISTINCT(TRIM(M_CAST.PID)) FROM M_CAST WHERE TRIM(M_
CAST.MID) IN
               (SELECT TRIM(MID) FROM M_CAST WHERE TRIM(PID) = (SELECT TRIM(PID) FROM PERSON WHERE PERSO
N.NAME LIKE '%Shah Rukh Khan%'))
                AND TRIM(M_CAST.PID) !='nm0451321') AND TRIM(M_CAST.PID) != (SELECT TRIM(PID) FROM PERSO
N WHERE PERSON.NAME LIKE '%Shah Rukh Khan%')) """
grader_9(query9)
```

```
              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)
CPU times: user 267 ms, sys: 11 ms, total: 278 ms
Wall time: 280 ms
```

In [ ]:

```
```