SQL Assignment

```
import pandas as pd
import sqlite3

from IPython.display import display, HTML
```

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

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
conn = sqlite3.connect("/content/drive/MyDrive/AI-ML-Assignments/SQL Assignment/Db-IMDB-As
```

Overview of all tables

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

```
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

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.

```
%%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 as Director,m.title as Movie,CAST(SUBSTR(TRIM(m.year),-4) AS IN
             join M_DIRECTOR MD on p.PID = MD.PID
             join Movie m on MD.MID = m.MID
             join M_Genre MG ON m.MID = MG.MID
             join Genre g on MG.GID = g.GID
             where (
             CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER)%4=0 AND
             CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER)%100 != 0 OR
             CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER)%400=0) and g.Name like '%Comedy%'  "
grader_1(query1)
```

```
              Director                             Movie  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 73.5 ms, sys: 3.43 ms, total: 76.9 ms
     Wall time: 84.5 ms
```

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

```
q2 = """ Select P.Name from Person P where PID in
         (Select TRIM(PID) from M_Cast where MID in
         (Select TRIM(MID) from Movie m
         where TRIM(m.title)='Anand' and CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER)=1971 ))
print(pd.read_sql_query(q2,conn))
```

```
                    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
10     Brahm Bhardwaj
11       Gurnam Singh
12       Lalita Pawar
13        Durga Khote
14         Dara Singh
15      Johnny Walker
16          Moolchand
```

```
%%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 Person P where PID in
         (Select TRIM(PID) from M_Cast where MID in
         (Select TRIM(MID) from Movie m
         where TRIM(m.title)='Anand' and CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER)=1971 ))
grader_2(query2)
```

```
                 Name
0   Amitabh Bachchan
1      Rajesh Khanna
2      Sumita Sanyal
3         Ramesh Deo
4          Seema Deo
5     Asit Kumar Sen
6         Dev Kishan
7       Atam Prakash
8      Lalita Kumari
9             Savita
CPU times: user 26.7 ms, sys: 2.85 ms, total: 29.6 ms
Wall time: 29.2 ms
```

Q3 --- List all the actors who acted in a film before 1970 and in a film after 1990. (That is: < 1970 and > 1990.)

```
%%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 257 ms, sys: 13.6 ms, total: 271 ms
Wall time: 273 ms
```

```
%%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.name from person p where p.pid in

            (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)
```

```
                  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 298 ms, sys: 4.1 ms, total: 302 ms
Wall time: 300 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.

```
%%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) from M_Director where MID in
              (Select TRIM(MID) from Movie M) Group by PID"""
print(grader_4a(query_4a))
```

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

```
           PID  count(MID)
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 22.6 ms, sys: 0 ns, total: 22.6 ms
Wall time: 22.9 ms
```

```
%%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 AS Name, COUNT(*) as Movies_Directed from Person p
        JOIN M_Director d on p.PID = d.PID
        JOIN Movie m on d.MID = m.MID
        GROUP BY Name HAVING Movies_Directed >= 10 ORDER BY Movies_Directed DESC  """
grader_4(query4)
```

```
                    Name  Movies_Directed
0          David Dhawan               39
1          Mahesh Bhatt               36
2       Ram Gopal Varma               30
3          Priyadarshan               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 48.1 ms, sys: 0 ns, total: 48.1 ms
Wall time: 51 ms
```

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

```
q5 = """ select CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER) as YEAR, COUNT(*) AS Female_Cast_
        (select mc.mid from person p
         inner join m_cast mc on p.pid = trim(mc.pid) where p.gender = 'Male' group by mc.
print(pd.read_sql_query(q5,conn))
print(pd.read_sql_query(q5,conn).shape)
```

```
        YEAR  Female_Cast_Only_Movies
```

```
0  1939                        1
1  1999                        1
2  2000                        1
3  2018                        1
(4, 2)
```

```
%%time

# note that you don't need TRIM for person table

def grader_5aa(query_5aa):
    query_5aa = pd.read_sql_query(query_5aa,conn)
    print(query_5aa.head(10))
    return (query_5aa.shape == (8846,3))

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

query_5aa =""" select mc.mid, p.gender, count(*) from person p
         inner join m_cast mc on p.pid = trim(mc.pid) group by mc.mid,p.gender """

print(grader_5aa(query_5aa))

def grader_5ab(query_5ab):
    query_5ab = pd.read_sql_query(query_5ab,conn)
    print(query_5ab.head(10))
    return (query_5ab.shape == (3469, 3))

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

query_5ab =""" select mc.mid, p.gender, count(*) from person p
         inner join m_cast mc on p.pid = trim(mc.pid) where p.gender = 'Male' group by mc.

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 359 ms, sys: 3.63 ms, total: 363 ms
      Wall time: 364 ms
```

```
%%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 CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER) as YEAR, COUNT(*) AS Female_
        (select mc.mid from person p
         inner join m_cast mc on p.pid = trim(mc.pid) where p.gender = 'Male' group by mc.
grader_5a(query5a)
```

```
         YEAR  Female_Cast_Only_Movies
      0  1939                        1
      1  1999                        1
      2  2000                        1
      3  2018                        1
      CPU times: user 159 ms, sys: 3.38 ms, total: 163 ms
      Wall time: 160 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.

```
%%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 A.YEAR, (B.Female_Cast_Only_Movies * 1.0)/sum(A.Total_movies) as Perc
          from
          (select CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER) as YEAR, count(mid)  AS Total_m
          group by YEAR)A

          join

          (select CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER) as YEAR, COUNT(mid) AS Female_C
```

```
          (select mc.mid from person p
          inner join m_cast mc on p.pid = trim(mc.pid) where p.gender = 'Male' group by mc

          on A.YEAR = B.YEAR  group by A.YEAR
          """
grader_5b(query5b)
```

```
   YEAR  Percentage_of_Female_cast_movies  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 156 ms, sys: 2.5 ms, total: 159 ms
Wall time: 160 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.

```
%%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 title as Name,Cast_ as Cast from

        (select m.title as title ,
        count(distinct(c.PID)) as Cast_
        from Movie m
        join M_cast c on c.MID = m.MID
        GROUP BY c.MID) order by cast_ desc """
grader_6(query6)
```

```
                          Name  Cast
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 189 ms, sys: 15.4 ms, total: 204 ms
Wall time: 206 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

```
%%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 CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER) as Movie_Year, Count(MID) as Tot
          """
grader_7a(query7a)

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

```
   Movie_Year  Total_Movies
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 11.9 ms, sys: 0 ns, total: 11.9 ms
Wall time: 12.1 ms
```

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

#Write a query that will do joining of the above table(7a) with itself
#such that you will join with only rows if the second tables year is <= current_year+9 and
```

```
query7b = """
            SELECT A.Movie_Year as First_year, A.Total_Movies as Movie_list_1, B.Movie_Yea
            (select CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER) as Movie_Year, Count(MID) as
            join
            (select CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER) as Movie_Year, Count(MID) as
            on A.Movie_Year <= B.Movie_Year and B.Movie_Year <= A.Movie_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
```

```
   First_year  Movie_list_1  Second_year  Movie_list_2
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 15.6 ms, sys: 988 µs, total: 16.5 ms
Wall time: 18.1 ms
```

```
%%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 count(*) as Total_Movies_in_decade , cast(m1.year as char)||'-'||cast(m1.ye
        (Select distinct(CAST(SUBSTR(YEAR,-4) AS INTEGER)) as year from Movie m)m1
        join
        (Select CAST(SUBSTR(YEAR,-4) AS INTEGER) as year from Movie m)m2
        on m1.year <= m2.year
        and m2.year <= m1.year+9 group by m1.year+9 order by 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 p
```

```
   Total_Movies_in_decade     Decade
0                    1203  2008-2017
CPU times: user 83.2 ms, sys: 0 ns, total: 83.2 ms
Wall time: 95.3 ms
```

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

```
 %%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 together **

query8a = """
          Select A.ID AS ACTOR_ID, B.ID AS DIRECTOR_ID, COUNT(*) AS MOVIES_MADE_WITH_DIREC
          (SELECT MC.MID AS MID, MC.PID AS ID FROM M_CAST MC JOIN PERSON P ON TRIM(MC.PID)
          JOIN
          (SELECT MD.MID AS MID, MD.PID AS ID FROM M_DIRECTOR MD JOIN PERSON P ON TRIM(MD.
          ON A.MID = B.MID GROUP BY ACTOR_ID, DIRECTOR_ID

          """
grader_8a(query8a)

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

```
     ACTOR_ID DIRECTOR_ID  MOVIES_MADE_WITH_DIRECTOR
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 478 ms, sys: 16.3 ms, total: 494 ms
Wall time: 494 ms
```

```
%%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))

#*** Write a query that answers the 8th question ***

query8 = """SELECT TRIM(Q1.ACTOR_ID) as ACTOR, Q1.MOVIES_MADE FROM
(Select A.ID AS ACTOR_ID, B.ID AS DIRECTOR_ID, COUNT(*) AS MOVIES_MADE FROM
(SELECT MC.MID AS MID, MC.PID AS ID FROM M_CAST MC JOIN PERSON P ON TRIM(MC.PID) = P.PID)A
JOIN
(SELECT MD.MID AS MID, MD.PID AS ID FROM M_DIRECTOR MD JOIN PERSON P ON TRIM(MD.PID) = P.P
ON A.MID = B.MID GROUP BY ACTOR_ID, DIRECTOR_ID)Q1

WHERE (Q1.ACTOR_ID, Q1.MOVIES_MADE)IN
```

```
(SELECT Q2.ACTOR_ID, MAX(MOVIES_MADE) FROM
(Select A.ID AS ACTOR_ID, B.ID AS DIRECTOR_ID, COUNT(*) AS MOVIES_MADE FROM
(SELECT MC.MID AS MID, MC.PID AS ID FROM M_CAST MC JOIN PERSON P ON TRIM(MC.PID) = P.PID)A
JOIN
(SELECT MD.MID AS MID, MD.PID AS ID FROM M_DIRECTOR MD JOIN PERSON P ON TRIM(MD.PID) = P.P
ON A.MID = B.MID GROUP BY ACTOR_ID, DIRECTOR_ID)Q2
GROUP BY Q2.ACTOR_ID)

AND Q1.DIRECTOR_ID = 'nm0007181'

ORDER BY MOVIES_MADE DESC









 """
grader_8(query8)
```

```
            ACTOR  MOVIES_MADE
     0  nm0707271           11
     1  nm0471443           10
     2  nm0407002            9
     3  nm0004434            7
     4  nm0347901            5
     5  nm0716851            5
     6  nm0433945            4
     7  nm0755087            4
     8  nm0802183            4
     9  nm0158332            3
     (245, 2)
     CPU times: user 620 ms, sys: 13.2 ms, total: 633 ms
     Wall time: 639 ms
```

```
q8 = """
     SELECT TRIM(Q1.ACTOR_ID), Q1.MOVIES_MADE FROM
(Select A.ID AS ACTOR_ID, B.ID AS DIRECTOR_ID, COUNT(*) AS MOVIES_MADE FROM
(SELECT MC.MID AS MID, MC.PID AS ID FROM M_CAST MC JOIN PERSON P ON TRIM(MC.PID) = P.PID)A
JOIN
(SELECT MD.MID AS MID, MD.PID AS ID FROM M_DIRECTOR MD JOIN PERSON P ON TRIM(MD.PID) = P.P
ON A.MID = B.MID GROUP BY ACTOR_ID, DIRECTOR_ID)Q1

WHERE (Q1.ACTOR_ID, Q1.MOVIES_MADE)IN


(SELECT Q2.ACTOR_ID, MAX(MOVIES_MADE) FROM
(Select A.ID AS ACTOR_ID, B.ID AS DIRECTOR_ID, COUNT(*) AS MOVIES_MADE FROM
(SELECT MC.MID AS MID, MC.PID AS ID FROM M_CAST MC JOIN PERSON P ON TRIM(MC.PID) = P.PID)A
JOIN
```

```
(SELECT MD.MID AS MID, MD.PID AS ID FROM M_DIRECTOR MD JOIN PERSON P ON TRIM(MD.PID) = P.P
ON A.MID = B.MID GROUP BY ACTOR_ID, DIRECTOR_ID)Q2
GROUP BY Q2.ACTOR_ID)

AND Q1.DIRECTOR_ID = 'nm0007181'

ORDER BY MOVIES_MADE DESC
"""

q8_results  = pd.read_sql_query(q8,conn)
print(q8_results)
```

```
    TRIM(Q1.ACTOR_ID)  MOVIES_MADE
0           nm0707271           11
1           nm0471443           10
2           nm0407002            9
3           nm0004434            7
4           nm0347901            5
..                ...          ...
240         nm7150152            1
241         nm7624361            1
242         nm7768629            1
243         nm8533115            1
244         nm8737993            1

[245 rows x 2 columns]
```

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.

```
%%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))

#Actors with shahrukh number 1

query9a = """ SELECT DISTINCT PID
  FROM M_CAST
 WHERE MID IN (
```

```
           SELECT TRIM(MID)
             FROM M_CAST
            WHERE TRIM(PID) IN (
                      SELECT TRIM(PID)
                        FROM PERSON P
                       WHERE TRIM(NAME) LIKE '%Shah Rukh Khan%'
                  )
       )
AND
       TRIM(PID) NOT IN (
           SELECT PID
             FROM PERSON P
            WHERE TRIM(NAME) LIKE '%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 actor
# removing S1 actors from the combined list of S1 & S2 actors, so that we get only S2 acto
```

```
              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 61.9 ms, sys: 1.48 ms, total: 63.4 ms
     Wall time: 65 ms
```

```
%%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 WHERE PID IN
(
SELECT DISTINCT TRIM(PID) AS ACTOR FROM M_CAST WHERE MID IN
    (SELECT DISTINCT MID FROM M_CAST WHERE TRIM(PID) IN
        (SELECT DISTINCT TRIM(PID) FROM M_CAST WHERE MID IN
            (SELECT TRIM(MID) FROM M_CAST WHERE TRIM(PID) IN
                (SELECT TRIM(PID) FROM PERSON P WHERE TRIM(NAME) LIKE '%Shah Rukh Khan%'))
            AND TRIM(PID) NOT IN
                (SELECT PID FROM PERSON P WHERE TRIM(NAME) LIKE '%Shah Rukh Khan%'))) AND
except
```

```
SELECT DISTINCT TRIM(PID) FROM M_CAST WHERE MID IN
            (SELECT TRIM(MID) FROM M_CAST WHERE TRIM(PID) IN
                (SELECT TRIM(PID) FROM PERSON P WHERE TRIM(NAME) LIKE '%Shah Rukh Khan%'))
            AND TRIM(PID) NOT IN
                (SELECT PID FROM PERSON P WHERE TRIM(NAME) LIKE '%Shah Rukh Khan%')
)"""
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)
CPU times: user 321 ms, sys: 18.6 ms, total: 340 ms
Wall time: 338 ms
```

✓ 0s completed at 7:35 PM