Copy of SQL sample queries

July 31, 2022

1 SQL Assignment

[1]:
```
import pandas as pd
import sqlite3

from IPython.display import display, HTML

conn = sqlite3.connect("/Users/srujan/Downloads/18.SQL/assign/
 ↪Db-IMDB-Assignment.db")
```

Overview of all tables

[2]:
```
tables = pd.read_sql_query("SELECT NAME AS 'Table_Name' FROM sqlite_master␣
 ↪WHERE type='table'",conn)
tables = tables["Table_Name"].values.tolist()
```

[3]:
```
tables
```

[3]:
```
['Movie',
 'Genre',
 'Language',
 'Country',
 'Location',
 'M_Location',
 'M_Country',
 'M_Language',
 'M_Genre',
 'Person',
 'M_Producer',
 'M_Director',
 'M_Cast']
```

[4]:
```
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
--------------------------------------------------------------------------------
--------------------
```

1.1 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 shown, 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(*)

[ ]:

1.2 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.

[5]:
```
%%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,year from M_Director md join M_Genre mg on␣
 ↪md.MID=mg.MID join Genre g on mg.GID=g.GID
join Movie m on m.MID=md.MID join person p on p.PID=md.PID
where g.Name like '%comedy%'
and CAST(SUBSTR(m.year,-4) AS Integer)%4=0 and (CAST(SUBSTR(m.year,-4) AS␣
 ↪Integer)%100!=0 or CAST(SUBSTR(m.year,-4) AS Integer)%400=0)
 """
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 87.5 ms, sys: 17.3 ms, total: 105 ms
Wall time: 121 ms
```

[ ]:

[ ]:

[ ]:

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

[6]:
```
%%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 (select trim(M_cast.PID) PID from M_cast where␣
 ↪(M_cast.MID) in (select trim(Movie.MID) from Movie where Movie.
 ↪title=\'Anand\')) as t1
                    left join Person
                    on t1.PID=Person.PID """
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 84.1 ms, sys: 16.5 ms, total: 101 ms
Wall time: 145 ms
```

[ ]:

[ ]:

[ ]:

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

[7]:
```
%%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 348 ms, sys: 16.2 ms, total: 365 ms
Wall time: 370 ms
```

[ ]:

[8]:
```
%%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 Name from Person where PID in
(SELECT ab
FROM
    (Select distinct(p.PID) ab 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) t1
INNER JOIN
    (Select distinct(p.PID) bc 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) t2
ON (ab = bc)); """
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 352 ms, sys: 11.3 ms, total: 364 ms
Wall time: 366 ms
```

[ ]:

[ ]:

[ ]:

1.5 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.

[9]:
```
%%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 =""" *** Write a query, which will return all the directors(id's)␣
 ↪along with the number of movies they directed *** """
query_4a =""" select PID as Director_ID,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
```

```
  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 11.8 ms, sys: 2.67 ms, total: 14.5 ms
Wall time: 12.6 ms
```

[ ]:
```

```

[10]:
```
%%time
def grader_4(q4):
    q4_results  = pd.read_sql_query(q4,conn)
    print(q4_results.head(10))
    assert (q4_results.shape == (58,2))

query4 = """ select Name,Movie_Count from
(select PID as Director_ID,count(MID) Movie_Count from M_Director group by PID␣
 ↪having Movie_Count>=10) t1
inner join Person
on (Director_ID=Person.PID) 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 21.8 ms, sys: 2.89 ms, total: 24.7 ms
Wall time: 22.8 ms
```

[ ]:

[ ]:

[ ]:

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

[11]:
```
%%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 DISTINCT trim(mc.MID) MD, p.Gender Gend, COUNT(*) Count \
                FROM M_Cast mc \
                JOIN Person p ON trim(mc.PID) = p.PID \
                GROUP BY MD,Gend """

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 DISTINCT trim(mc.MID) MD, p.Gender Gend, COUNT(*) Count
                FROM M_Cast mc
                JOIN Person p ON trim(mc.PID) = p.PID
                GROUP BY MD,Gend having Gend=\"Male\" """

print(grader_5ab(query_5ab))

45
# using the above queries, you can write the answer to the given question
```

```
          MD    Gend  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
          MD  Gend  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 409 ms, sys: 41 ms, total: 450 ms
Wall time: 461 ms
```

[11]:
```
45
```

[ ]:
```
```

[12]:
```
%%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,Female_Cast_Only_Movies from (select MD,count(*)␣
 ↪Female_Cast_Only_Movies,Gend from (SELECT DISTINCT trim(mc.MID) MD, p.Gender␣
 ↪Gend, COUNT(*) Count
FROM M_Cast mc
JOIN Person p ON trim(mc.PID) = p.PID
GROUP BY MD,Gend) group by MD having Gend = \"Female\" and␣
 ↪Female_Cast_Only_Movies=1)
join Movie m
on m.MID = MD order by year """
grader_5a(query5a)
```

```
     year  Female_Cast_Only_Movies
0    1939                        1
1    1999                        1
2    2000                        1
3  I 2018                        1
CPU times: user 249 ms, sys: 19.9 ms, total: 269 ms
Wall time: 279 ms
```

[ ]:

[ ]:

[ ]:

1.7 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.

[13]:
```
%%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 year, CAST(Female_Cast_Only_Movies AS float) /␣
 ↪CAST(Total_Movies AS float) Percentage_Female_Only_Movie,Total_Movies  from
(select MD,Female_Cast_Only_Movies,Gend, CAST(SUBSTR(m.year,-4) AS Integer)␣
 ↪year from (select MD,count(*) Female_Cast_Only_Movies,Gend from (SELECT␣
 ↪DISTINCT trim(mc.MID) MD, p.Gender Gend, COUNT(*) Count
FROM M_Cast mc
JOIN Person p ON trim(mc.PID) = p.PID
GROUP BY MD,Gend) group by MD having Gend = \"Female\" and␣
 ↪Female_Cast_Only_Movies=1)
```

```
join Movie m
on trim(m.MID)=MD) inner join
(select CAST(SUBSTR(Movie.year,-4) AS Integer) year_,count(*) Total_Movies from␣
 ↪Movie group by year_)
on year=year_ order by 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 240 ms, sys: 18.1 ms, total: 258 ms
Wall time: 282 ms
```

[ ]:

[ ]:

[ ]:

1.8 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.

[14]:
```
%%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,count from
(select trim(mc.MID) mcmid ,count(distinct(PID)) count from M_Cast mc group by␣
 ↪mcmid order by count desc)
join Movie m
on m.MID = mcmid """
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 105 ms, sys: 20 ms, total: 125 ms
Wall time: 131 ms
```

[ ]:

[ ]:

[ ]:

1.8.1 Q7 — A decade is a sequence of 10 consecutive years.

1.8.2 For example, say in your database you have movie information starting from 1931.

1.8.3 the first decade is 1931, 1932, ..., 1940,

1.8.4 the second decade is 1932, 1933, ..., 1941 and so on.

1.8.5 Find the decade D with the largest number of films and the total number of films in D

[15]:
```
%%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 CAST(SUBSTR(m.year,-4) AS Integer) year_,␣
 ↪count((trim(m.MID))) MID from Movie m group by year_ """
grader_7a(query7a)

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

```
   year_  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
CPU times: user 7.39 ms, sys: 2.03 ms, total: 9.42 ms
Wall time: 7.99 ms
```

[ ]:

[16]:
```
%%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 * from
(select  distinct CAST(SUBSTR(m.year,-4) AS Integer) year_, count((trim(m.
 ↪MID))) MID from Movie m group by year_)
join (select  distinct CAST(SUBSTR(m.year,-4) AS Integer) year__, count((trim(m.
 ↪MID))) MID_ from Movie m group by year__)
on (year__<=year_+9 and year__>=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
```

```
   year_  MID  year__  MID_
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 15.5 ms, sys: 2.61 ms, total: 18.1 ms
Wall time: 17.6 ms
```

[ ]:

[17]:
```
%%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 max(s) Decade_Movie_Count,year_ Decade from
```

```
(select year_,count(*) c,sum(MID_) s from
(select  distinct CAST(SUBSTR(m.year,-4) AS Integer) year_, count((trim(m.
 ↪MID))) MID from Movie m group by year_)
join (select  distinct CAST(SUBSTR(m.year,-4) AS Integer) year__, count((trim(m.
 ↪MID))) MID_ from Movie m group by year__)
on (year__<=year_+9 and year__>=year_) group by year_ having c>=10) """
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_Movie_Count  Decade
0                1203    2008
CPU times: user 11.3 ms, sys: 2.37 ms, total: 13.6 ms
Wall time: 11.9 ms
```

[ ]:

[ ]:

[ ]:

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

[18]:
```
%%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 mc.PID Actors,md.PID Directors,count(*) Movies from␣
 ↪M_Director md join M_Cast mc on md.MID=mc.MID group by md.PID,mc.PID"""
grader_8a(query8a)

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

```
      Actors  Directors  Movies
0  nm0000027  nm0000180       1
1  nm0001114  nm0000180       1
2  nm0001919  nm0000180       1
3  nm0006762  nm0000180       1
4  nm0030062  nm0000180       1
5  nm0038970  nm0000180       1
6  nm0051856  nm0000180       1
```

```
7   nm0085966  nm0000180       1
8   nm0097889  nm0000180       1
9   nm0125497  nm0000180       1
CPU times: user 430 ms, sys: 50.9 ms, total: 481 ms
Wall time: 519 ms
```

[ ]:

[19]:
```
%%time

#https://stackoverflow.com/questions/62852386/
 ↪find-all-the-actors-that-made-more-movies-with-yash-chopra-than-any-other-direct␣
 ↪

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 Name,Movies from
(select Actors,Movies from
(select trim(mc.pid) as Actors,
        md.pid as Directors,
        count(*) as Movies,
        rank() over (partition by mc.pid order by count(*) desc) as rn
  from m_director as md
  join
  m_cast as mc on md.mid = mc.mid
  left join
  person as p on md.pid=p.pid and name = 'Yash Chopra'
  group by mc.pid,md.pid) where rn =1 and Directors in (select PID from Person␣
 ↪where Name like '%yash chopra'))
left join person p on Actors = p.PID order by Movies desc"""
grader_8(query8)
```

```
                Name  Movies
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 655 ms, sys: 54.6 ms, total: 710 ms
Wall time: 724 ms
```

[ ]:

[ ]:

[ ]:

1.10 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.

[20]:
```
%%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 S1_PID from
(select distinct(trim(PID)) S1_PID from
(select distinct(trim(MID)) srk_mid from M_Cast where trim(PID) = (select PID␣
 ↪from Person where Name like '%Shah Rukh Khan')) join M_Cast mc on trim(mc.
 ↪MID) = srk_mid)
where not S1_PID = 'nm0451321'

"""
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 73.5 ms, sys: 6.88 ms, total: 80.3 ms
Wall time: 81.7 ms
```

[ ]:

[21]:
```
%%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 Name from
(select * from
(select * from
(select distinct(trim(PID)) s2_actors from
(select distinct(trim(MID)) s2_movies from
(select * from
(select distinct(trim(PID)) s1_actors from
(select distinct(trim(MID)) srk_mid from M_Cast where trim(PID) = (select PID␣
 ↪from Person where Name like '%Shah Rukh Khan')) join M_Cast mc on trim(mc.
 ↪MID) = srk_mid)
where not s1_actors = 'nm0451321') join M_Cast on trim(M_Cast.PID) = s1_actors)
join M_Cast mc on trim(mc.MID)=s2_movies
) where s2_actors not in
(select S1_PID from
(select distinct(trim(PID)) S1_PID from \
(select distinct(trim(MID)) srk_mid from M_Cast where trim(PID) = (select PID␣
 ↪from Person where Name like '%Shah Rukh Khan')) join M_Cast mc on trim(mc.
 ↪MID) = srk_mid)
where not S1_PID = 'nm0451321'
))
where not s2_actors = 'nm0451321')
join Person p on p.PID=s2_actors"""
grader_9(query9)
```

```
                  Name
0       Alicia Vikander
1          Dominic West
2        Walton Goggins
3             Daniel Wu
```

```
4   Kristin Scott Thomas
5           Derek Jacobi
6     Alexandre Willaume
7           Tamer Burjaq
8         Adrian Collins
9         Keenan Arrison
(25698, 1)
CPU times: user 446 ms, sys: 15.5 ms, total: 462 ms
Wall time: 463 ms
```

[ ]:

[ ]:

[ ]: