SQL_Assignment

February 9, 2021

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

from IPython.display import display, HTML
```

[2]:
```
!gdown https://drive.google.com/uc?id=1O-1-L1DdNxEK6O6nG2jS31MbrMh-OnXM
```

```
Downloading…
From: https://drive.google.com/uc?id=1O-1-L1DdNxEK6O6nG2jS31MbrMh-OnXM
To: /content/Db-IMDB-Assignment.db
7.51MB [00:00, 97.7MB/s]
```

[3]:
```
%load_ext sql
%config SqlMagic.autocommit=True # for engines that do not support autommit

%sql sqlite:///Db-IMDB-Assignment.db
```

[3]:
```
'Connected: @Db-IMDB-Assignment.db'
```

[4]:
```
%%time
%%sql

UPDATE Person SET name = RTRIM(LTRIM(name));
UPDATE Person SET pid = RTRIM(LTRIM(pid));
UPDATE Person SET gender = RTRIM(LTRIM(gender));

UPDATE M_Cast SET pid = RTRIM(LTRIM(pid));
UPDATE M_Cast SET mid = RTRIM(LTRIM(mid));

UPDATE Movie SET year = REPLACE(year, "I", "");
UPDATE Movie SET year = REPLACE(year, "V", "");
UPDATE Movie SET year = REPLACE(year, "X ", "");
UPDATE Movie SET title = LTRIM(title);
UPDATE Movie SET year = RTRIM(LTRIM(year));
UPDATE Movie SET rating = RTRIM(LTRIM(rating));
UPDATE Movie SET num_votes = RTRIM(LTRIM(num_votes));

UPDATE M_Producer SET pid = RTRIM(LTRIM(pid));
UPDATE M_Producer SET mid = RTRIM(LTRIM(mid));

UPDATE M_Genre SET gid = RTRIM(LTRIM(gid));
UPDATE M_Genre SET mid = RTRIM(LTRIM(mid));

UPDATE Genre SET gid = RTRIM(LTRIM(gid));
UPDATE Genre SET name = RTRIM(LTRIM(name));

UPDATE M_Director SET pid = RTRIM(LTRIM(pid));
UPDATE M_Director SET mid = RTRIM(LTRIM(mid));
```

```
 * sqlite:///Db-IMDB-Assignment.db
37566 rows affected.
37566 rows affected.
37566 rows affected.
82835 rows affected.
82835 rows affected.
3473 rows affected.
3473 rows affected.
3473 rows affected.
3473 rows affected.
3473 rows affected.
3473 rows affected.
3473 rows affected.
11749 rows affected.
11749 rows affected.
3473 rows affected.
3473 rows affected.
328 rows affected.
328 rows affected.
3473 rows affected.
3473 rows affected.
CPU times: user 176 ms, sys: 61.6 ms, total: 238 ms
Wall time: 644 ms
```

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

Overview of all tables

[5]:
```
conn = sqlite3.connect("Db-IMDB-Assignment.db")

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

[6]:
```
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
```


0.1 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 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(*)

0.2 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.

[7]:
```
%%time

def grader_1(q1):
    q1_results  = pd.read_sql_query(q1,conn)
    print(q1_results.shape)
    print(q1_results.head(10))
    assert (q1_results.shape == (232,3))

query1 = '''SELECT
p.Name as Director_Name,
m.title as Movie_Name,
m.year as Year
FROM Movie as m
JOIN M_Director as md on md.MID = m.MID
JOIN M_Genre as mg on md.MID = mg.MID
JOIN Genre as g on g.GID = mg.GID
JOIN Person as p on md.PID = TRIM(p.PID)
```

```
WHERE (year % 4 = 0 AND year % 100 != 0 OR year % 400 = 0) AND (g.Name LIKE␣
↪'%Comedy%')
ORDER BY year ASC
'''
grader_1(query1)
```

```
(232, 3)
       Director_Name       Movie_Name  Year
0       Chetan Anand         Funtoosh  1956
1        Mohan Segal        New Delhi  1956
2         Amit Mitra       Jagte Raho  1956
3        Satyen Bose          Jagriti  1956
4          Bimal Roy           Parakh  1960
5        R.K. Nayyar    Love in Simla  1960
6         S.U. Sunny         Kohinoor  1960
7  Kidar Nath Sharma      Chitralekha  1964
8     Shakti Samanta  Kashmir Ki Kali  1964
9      Ravindra Dave     Dulha Dulhan  1964
CPU times: user 48.6 ms, sys: 6.11 ms, total: 54.7 ms
Wall time: 60.3 ms
```

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

[8]:
```
%%time

def grader_2(q2):
    q2_results  = pd.read_sql_query(q2,conn)
    print(q2_results.head(10))
    assert (q2_results.shape == (17,1))


query2 = '''SELECT p.Name
FROM Person p
JOIN M_Cast mc ON p.PID = TRIM(mc.PID)
JOIN Movie m ON m.MID = mc.MID
WHERE m.title = 'Anand' AND m."year" = 1971'''

grader_2(query2)
```

```
               Name
0  Amitabh Bachchan
1     Rajesh Khanna
2    Brahm Bhardwaj
3        Ramesh Deo
4         Seema Deo
5        Dev Kishan
6       Durga Khote
```

```
7     Lalita Kumari
8      Lalita Pawar
9      Atam Prakash
CPU times: user 149 ms, sys: 9.5 ms, total: 159 ms
Wall time: 161 ms
```

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

[9]:
```
%%time
# %%sql

query = '''WITH
CAST_ON_1970 AS (SELECT TRIM(p.PID) FROM Person p JOIN M_Cast mc ON p.PID =␣
↪TRIM(mc.PID) JOIN Movie m ON m.MID = mc.MID WHERE m.year < 1970),
CAST_ON_1990 AS (SELECT TRIM(p.PID) FROM Person p JOIN M_Cast mc ON p.PID =␣
↪TRIM(mc.PID) JOIN Movie m ON m.MID = mc.MID WHERE m.year > 1990)
SELECT p.Name AS 'Actor'
FROM Person p
WHERE TRIM(PID) IN CAST_ON_1970 AND TRIM(p.PID) IN CAST_ON_1990
'''
```

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

[10]:
```
%%time
def grader_3(q3):
    q3_results  = pd.read_sql_query(q3,conn)
    print(q3_results.shape)
    print(q3_results.head(10))
    assert (q3_results.shape == (300,1))

query3 = query
grader_3(query3)
```

```
(300, 1)
               Actor
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 409 ms, sys: 8.3 ms, total: 418 ms
```

```
Wall time: 420 ms
```

0.5 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.

[11]:
```
query = '''WITH
PID_FREQ AS (SELECT md.PID, COUNT(*) as movies_count FROM M_Director md GROUP␣
 ↪BY md.PID HAVING COUNT(*) >= 10)
SELECT p.Name as directors,
pf.movies_count as no_of_directed_movies
from Person p
JOIN PID_FREQ as pf ON pf.PID = p.PID ORDER BY no_of_directed_movies DESC'''
```

[12]:
```
%%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 = query
grader_4(query4)
```

```
             directors  no_of_directed_movies
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 24.4 ms, sys: 1.25 ms, total: 25.7 ms
Wall time: 27.5 ms
```

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

[13]:
```
query = '''WITH
NON_FEMALE_MOVIES AS (SELECT TRIM(MC.MID) FROM M_Cast MC INNER JOIN Person P ON␣
 ↪P.PID = TRIM(MC.PID) WHERE P.Gender in ('Male', NULL) GROUP BY MC.MID),
FEMALE_MOVIES AS (SELECT M.MID FROM Movie M INNER JOIN M_Cast MC ON TRIM(MC.
 ↪MID) = M.MID WHERE TRIM(M.MID) NOT IN NON_FEMALE_MOVIES AND MC.PID NOTNULL␣
 ↪GROUP BY M.MID)
```

```
SELECT M.year, COUNT(*) AS 'Female_Cast_Only_Movies'
FROM Movie M
WHERE TRIM(M.MID) IN FEMALE_MOVIES
GROUP BY M.year
ORDER BY M.year'''

pd.read_sql_query(query,conn)
```

[13]:
```
   year  Female_Cast_Only_Movies
0  1939                        1
1  1999                        1
2  2000                        1
3  2018                        1
```

[14]:
```
%%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 = query
grader_5a(query5a)
```

```
   year  Female_Cast_Only_Movies
0  1939                        1
1  1999                        1
2  2000                        1
3  2018                        1
CPU times: user 238 ms, sys: 2.01 ms, total: 240 ms
Wall time: 241 ms
```

0.7 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.

[15]:
```
query = '''WITH
NON_FEMALE_MOVIES AS (SELECT TRIM(MC.MID) FROM M_Cast MC INNER JOIN Person P ON␣
 ↪P.PID = TRIM(MC.PID) WHERE P.Gender in ('Male', NULL) GROUP BY MC.MID),
FEMALE_MOVIES AS (SELECT M.MID FROM Movie M INNER JOIN M_Cast MC ON TRIM(MC.
 ↪MID) = M.MID WHERE TRIM(M.MID) NOT IN NON_FEMALE_MOVIES AND MC.PID NOTNULL␣
 ↪GROUP BY M.MID),
ALL_YEARS AS (SELECT m.year, COUNT(*) AS 'total_movies' FROM Movie as m GROUP␣
 ↪BY m.year)
```

```
SELECT m.year, AY.total_movies, (COUNT(m.year) * 100 * 0.01 / AY.total_movies)␣
 ↪AS Percentage_Female_Only_Movie
FROM Movie m
INNER JOIN FEMALE_MOVIES FM ON FM.MID = m.MID
INNER JOIN ALL_YEARS AY ON m.year = AY.year
GROUP BY m.year
ORDER BY m.year '''
```

[16]:
```
%%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 = query
grader_5b(query5b)
```

```
   year  total_movies  Percentage_Female_Only_Movie
0  1939             2                      0.500000
1  1999            66                      0.015152
2  2000            64                      0.015625
3  2018           104                      0.009615
CPU times: user 240 ms, sys: 6.61 ms, total: 247 ms
Wall time: 250 ms
```

0.8 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.

[17]:
```
query = '''WITH
CAST_COUNT AS
(SELECT COUNT(*) AS CAST_COUNT, mc.MID
FROM M_Cast mc
GROUP BY mc.MID
ORDER BY CAST_COUNT DESC)

SELECT M.title as MOVIE_TITLE, CC.CAST_COUNT AS CAST_SIZE FROM CAST_COUNT CC
JOIN Movie AS M ON M.MID = CC.MID
ORDER BY CAST_SIZE DESC
'''
```

[18]:
```
%%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 = query
grader_6(query6)
```

```
                 MOVIE_TITLE  CAST_SIZE
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 57.3 ms, sys: 1.77 ms, total: 59 ms
Wall time: 59.9 ms
```

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

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

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

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

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

[19]:
```
query = '''WITH
UNIQUE_YEAR AS (SELECT DISTINCT year FROM Movie)

SELECT D.year as DECADE_START, D.year+9 as DECADE_END, COUNT(*) AS␣
 ↪'TOTAL_FILMS'
FROM UNIQUE_YEAR D
JOIN Movie M on M.year >= D.year AND M.year<= D.year+9
GROUP BY D.year+9
ORDER BY COUNT(*) DESC LIMIT 1;'''

pd.read_sql_query(query, conn)
```

[19]:
```
   DECADE_START  DECADE_END  TOTAL_FILMS
0          2008        2017         1203
```

[20]:
```
%%time
def grader_7(q7):
    q7_results  = pd.read_sql_query(q7,conn)
    print(q7_results.head(10))
    assert (q7_results.shape == (1, 3))
```

```
grader_7(query)
# if you check the output we are printinng all the year in that decade, its␣
↪fine you can print 2008 or 2008-2017
```

```
  DECADE_START  DECADE_END  TOTAL_FILMS
0         2008        2017         1203
CPU times: user 71.6 ms, sys: 818 µs, total: 72.5 ms
Wall time: 73.2 ms
```

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

[21]:
```
query = '''SELECT TRIM(P.PID) AS ACTOR_NAME,
TRIM(P1.PID) AS DIRECTOR_NAME,
COUNT(DISTINCT M.MID) AS MOVIES_COUNTS

FROM Person P

JOIN M_Cast MC ON TRIM(MC.PID) = P.PID
JOIN Movie M ON M.MID = MC.MID
JOIN M_Director MD ON MD.MID = M.MID
JOIN Person P1 ON P1.PID = TRIM(MD.PID)

GROUP BY ACTOR_NAME, DIRECTOR_NAME
'''
```

[22]:
```
%%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 = query
grader_8a(query8a)

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

```
  ACTOR_NAME DIRECTOR_NAME  MOVIES_COUNTS
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 688 ms, sys: 32 ms, total: 720 ms
Wall time: 726 ms
```

0.9.1 REFERENCE HYPERLINK: STACKOVERFLOW

[23]:
```
query = '''select *
from m_cast as mc
join m_director md
on md.MID=mc.MID
group by mc.pid , md.pid

'''

pd.read_sql_query(query,conn).head(5)
# nm0007181
```

[23]:
```
   index        MID        PID     ID  index        MID        PID    ID
0  19391  tt0053126  nm0000002  19391    529  tt0053126  nm0496746   529
1   4696  tt0087892  nm0000027   4696     91  tt0087892  nm0000180    91
2  62041  tt0046427  nm0000039  62041   2380  tt0046427  nm0896533  2380
3  62040  tt0046427  nm0000042  62040   2380  tt0046427  nm0896533  2380
4  15894  tt0066070  nm0000047  15894    407  tt0066070  nm0004292   407
```

[24]:
```
query = '''select p.name, h.count
from(select mc.pid as mcpid, md.pid as mdpid, count(mc.MID) as count
from m_cast as mc
join m_director md
on md.MID=mc.MID
group by mc.pid , md.pid) h
join person p
on h.mcpid=p.pid
where h.count = (select count(*) as count
from m_cast as mc
join m_director md
on md.mid=mc.mid
where mc.pid=h.mcpid
group by mc.pid,md.pid
order by count(*) desc
limit 1)
and h.mdpid = (select pid
from person
where name like '%Yash Chopra%')
order by h.count desc
'''

pd.read_sql_query(query,conn).head(15)
# nm0007181
```

```
[24]:                  Name  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
      10       Surendra Rahi     3
      11        Sudha Chopra     3
      12         Saul George     3
      13        Mohan Sherry     3
      14       Leela Chitnis     3
```

```
[25]: %%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 = query
grader_8(query)
```

```
                Name  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 3.04 s, sys: 450 ms, total: 3.49 s
Wall time: 3.5 s
```

0.10 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.

[26]:
```
query = '''WITH
SHARUKH_0 AS (SELECT TRIM(PID) AS PID FROM Person P
where P.Name like 'Sha%Rukh%Khan'),

SHARUKH_MOVIES_1 AS (SELECT DISTINCT TRIM(MC.MID) MID, S0.PID
FROM M_Cast MC, SHARUKH_0 S0
WHERE TRIM(MC.PID) = S0.PID),

SHARUKH_1_ACTORS AS (SELECT DISTINCT TRIM(MC.PID) AS PID FROM M_cast MC,␣
 ↪SHARUKH_MOVIES_1 SM1
WHERE TRIM(MC.MID) = SM1.MID AND TRIM(MC.PID) <> SM1.PID),

SHARUKH_MOVIES_2 AS (SELECT DISTINCT TRIM(MC.MID) MID, S1.PID
FROM M_Cast MC, SHARUKH_1_ACTORS S1
WHERE TRIM(MC.PID) = S1.PID),

SET_ACTOR1_ACTOR2 AS (SELECT DISTINCT TRIM(MC.PID) AS PID FROM M_cast MC,␣
 ↪SHARUKH_MOVIES_2 SM2
WHERE TRIM(MC.MID) = SM2.MID AND TRIM(MC.PID) <> SM2.PID),

ACTORS_2 AS (SELECT * FROM SET_ACTOR1_ACTOR2 G
WHERE TRIM(G.PID) NOT IN (SELECT TRIM(PID) FROM SHARUKH_1_ACTORS)
ORDER BY G.PID)

SELECT P.Name FROM Person P
JOIN ACTORS_2 A2 ON TRIM(A2.PID) = P.PID
'''
```

[27]:
```
%%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 == (25699, 1))

query9 = query
grader_9(query9)
```

```
                 Name
0      Alec Guinness
1       Sophia Loren
```

```
2         Brad Pitt
3  Gillian Anderson
4    Pierce Brosnan
5          Bo Derek
6   Michael Douglas
7        Cary Elwes
8       Colin Firth
9   Whoopi Goldberg
(25699, 1)
CPU times: user 917 ms, sys: 14.8 ms, total: 932 ms
Wall time: 932 ms
```