SQL IMDB Queries

January 21, 2022

Let's `import` useful packages for this assignment.

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

Now, we need to connect to the IMDB database using `sqlite3.connect()` method.

[2]:
```
conn = sqlite3.connect(database="DB-IMDB-Assignment.db")
```

Below are the list of all the tables inside IMDB database.

[3]:
```
table_query = """
SELECT
    NAME AS "Table_Name"
FROM
    sqlite_master
WHERE
    type = "table"
"""
```

[4]:
```
tables = pd.read_sql_query(sql=table_query, con=conn)
display(tables)

tables = tables["Table_Name"].values.tolist()
```

	Table_Name
0	Movie
1	Genre
2	Language
3	Country
4	Location
5	M_Location
6	M_Country
7	M_Language
8	M_Genre
9	Person
10	M_Producer
11	M_Director
12	M_Cast

[5]:
```
for table in tables:
    query = "PRAGMA TABLE_INFO({})".format(table)
    schema = pd.read_sql_query(sql=query, con=conn)
    print("Schema of", table)
    display(schema)
```

```
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

Useful tips

1. The year column in 'Movie' table, will have few chracters other than numbers which needs to be preprocessed, you need to get a substring of last 4 characters, its better if you convert it as integer type, for example: `CAST(SUBSTR(TRIM(year),-4) AS INTEGER)`.
2. For almost all the `TEXT` columns we show, please try to remove trailing spaces, you need to use `TRIM()` function.
3. When you are doing `COUNT(column)` 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 the director name, the movie name, and the year.

To determine whether a year is a leap year or not, follow these steps:

1. If the year is evenly divisible by 4, go to step 2. Otherwise, go to step 5.
2. If the year is evenly divisible by 100, go to step 3. Otherwise, go to step 4.
3. If the year is evenly divisible by 400, go to step 4. Otherwise, go to step 5.
4. The year is a leap year (it has 366 days).
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.

First, let's preprocess Movie table's year column.

[6]:
```
query1 = """
SELECT
    P.Name Director,
    M.title Title,
    CAST(SUBSTRING(TRIM(M.year), -4) AS INTEGER) Year
FROM
    Person AS P
JOIN
    M_Director AS MD
ON
    P.PID = MD.PID
JOIN
    Movie AS M
ON
   MD.MID = M.MID
JOIN
    M_Genre AS MG
ON
    M.MID = MG.MID
JOIN
```

```
    Genre AS G
ON
    MG.GID = G.GID
WHERE
    G.Name LIKE "%Comedy%"
AND (
    CAST(SUBSTRING(TRIM(M.year), -4) AS INTEGER) % 4 = 0 AND
    CAST(SUBSTRING(TRIM(M.year), -4) AS INTEGER) % 100 <> 0 OR
    CAST(SUBSTRING(TRIM(M.year), -4) AS INTEGER) % 400 = 0
)
"""
```

[7]:
```
%%time

def grader_1(q1):
    q1_results  = pd.read_sql_query(sql=q1, con=conn)
    display(q1_results.head(n=10))
    assert (q1_results.shape == (232, 3))

grader_1(q1=query1)
```

	Director	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 44.4 ms, sys: 1.19 ms, total: 45.6 ms
Wall time: 53.7 ms
```

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

[8]:
```
query2 = """
SELECT
    P.Name Actor_Names
FROM
    Person AS P
JOIN
    M_Cast AS MC
ON
    P.PID = TRIM(MC.PID)
```

```
JOIN
    Movie AS M
ON
    TRIM(MC.MID) = TRIM(M.MID)
WHERE
    TRIM(M.title) = "Anand"
"""
```

[9]:
```
%%time

def grader_2(q2):
    q2_results  = pd.read_sql_query(sql=q2, con=conn)
    display(q2_results.head(n=10))
    assert (q2_results.shape == (17, 1))

grader_2(q2=query2)
```

```
        Actor_Names
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 41.9 ms, sys: 758 µs, total: 42.7 ms
Wall time: 53.2 ms
```

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

[10]:
```
q_1_1970 ="""
SELECT
    DISTINCT P.PID
FROM
    Person AS P
JOIN (
    SELECT
        TRIM(MC.PID) PID,
        MC.MID
    FROM
        M_Cast AS MC
    WHERE
```

```
        MC.MID IN (
            SELECT
                MV.MID
            FROM
                Movie AS MV
            WHERE
                CAST(SUBSTRING(MV.year, -4) AS INTEGER) < 1970
        )
) AS R1
ON
    R1.PID = P.PID
"""
```

[11]:
```
q_m_1990 ="""
SELECT
    DISTINCT P.PID
FROM
    Person AS P
JOIN (
    SELECT
        TRIM(MC.PID) PID,
        MC.MID
    FROM
        M_Cast AS MC
    WHERE
        MC.MID IN (
            SELECT
                MV.MID
            FROM
                Movie AS MV
            WHERE
                CAST(SUBSTRING(MV.year, -4) AS INTEGER) > 1990
        )
) AS R1
ON
    R1.PID = P.PID
"""
```

[12]:
```
%%time

def grader_3a(q_l_1970, q_m_1990):
    q3a = pd.read_sql_query(sql=q_l_1970, con=conn)
    display(q3a.head(n=5))
    print(q3a.shape)
    q3b = pd.read_sql_query(sql=q_m_1990, con=conn)
    display(q3b.head(n=5))
    print(q3b.shape)
```

```
    return q3a.shape == (4942, 1) and q3b.shape == (62570, 1)

grade_3a_ = grader_3a(q_l_1970=q_l_1970, q_m_1990=q_m_1990)
print(grade_3a_)
```

```
        PID
0  nm0719692
1  nm0623658
2  nm0549280
3  nm0415488
4  nm0336474

(1959, 1)

        PID
0  nm0000288
1  nm0000949
2  nm1212722
3  nm0365140
4  nm0785227

(27961, 1)
False
CPU times: user 154 ms, sys: 3.46 ms, total: 158 ms
Wall time: 157 ms
```

[13]:
```
query3 = """
WITH
    ql1970 AS ({}),
    qm1990 AS ({})
SELECT
    P.Name Actor_Name
FROM
    Person AS P
JOIN
    ql1970
ON
    TRIM(ql1970.PID) = P.PID
JOIN
    qm1990
ON
    P.PID = TRIM(qm1990.PID)
""".format(q_l_1970, q_m_1990)
```

[14]:
```
%%time

def grader_3(q3):
    q3_results  = pd.read_sql_query(sql=q3, con=conn)
    display(q3_results.head(n=10))
```

```
    assert (q3_results.shape == (300,1))

grader_3(q3=query3)
```

```
        Actor_Name
0   Waheeda Rehman
1    Johnny Walker
2          Mehmood
3            Ratna
4   Rajendra Kumar
5          Iftekhar
6        Raj Mehra
7     Lalita Pawar
8   Achala Sachdev
9       Sunil Dutt

CPU times: user 21 s, sys: 298 µs, total: 21 s
Wall time: 21.1 s
```

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.

a) Write a query, which will return all the directors(id's) along with the number of movies they directed.

[15]:
```
q_4a = """
SELECT
    MD.PID Director_ID,
    COUNT(M.title) Movie_Count
FROM
    M_Director AS MD
JOIN
    Movie AS M
ON
    TRIM(MD.MID) = TRIM(M.MID)
GROUP BY
    TRIM(MD.PID)
"""
```

[16]:
```
%%time

def grader_4a(q_4a):
    q4a = pd.read_sql_query(sql=q_4a, con=conn)
    display(q4a.head(n=10))
    return (q4a.shape == (1462, 2))

grader_4a_ = grader_4a(q_4a=q_4a)
print(grader_4a_)
```

```
  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 848 ms, sys: 982 µs, total: 849 ms
Wall time: 849 ms
```

[17]:
```
query4 = """
WITH
    q4a AS ({})
SELECT
    P.Name Director_Name,
    q4a.Movie_Count
FROM
    Person AS P
JOIN
    q4a
ON
    P.PID = TRIM(q4a.Director_ID)
WHERE
    q4a.Movie_Count >= 10
ORDER BY
    q4a.Movie_Count DESC
""".format(q_4a)
```

[18]:
```
%%time

def grader_4(q4):
    q4_results  = pd.read_sql_query(sql=q4, con=conn)
    display(q4_results.head(n=10))
    assert (q4_results.shape == (58, 2))

grader_4(q4=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 825 ms, sys: 0 ns, total: 825 ms
Wall time: 825 ms
```

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

a) Write your query that will get movie id, and number of people for each gender.

b) Write your query that will have at least one male actor try to use query that you have written above.

[19]:
```
q_5aa = """
SELECT
    TRIM(MC.MID) MID,
    TRIM(P.Gender) Gend,
    COUNT(P.Gender) Count
FROM
    M_Cast AS MC
JOIN
    Person AS P
ON
    TRIM(MC.PID) = P.PID
GROUP BY
    MID,
    Gend
"""
```

[20]:
```
q_5ab = """
WITH
    q5aa AS ({})
SELECT
    *
FROM
    q5aa
WHERE
    q5aa.Gend = "Male"
AND
    q5aa.Count >= 1
""".format(q_5aa)
```

[21]:
```
%%time

def grader_5aa(q_5aa):
```

```
    q5aa = pd.read_sql_query(sql=q_5aa, con=conn)
    display(q5aa.head(n=10))
    return (q5aa.shape == (8846, 3))

grader_5aa_ = grader_5aa(q_5aa=q_5aa)
print(grader_5aa_)

def grader_5ab(q_5ab):
    q5ab = pd.read_sql_query(sql=q_5ab, con=conn)
    display(q5ab.head(n=10))
    return (q5ab.shape == (3469, 3))

grader_5ab_ = grader_5ab(q_5ab=q_5ab)
print(grader_5ab_)
```

```
         MID    Gend  Count
0  tt0021594    None      0
1  tt0021594  Female      3
2  tt0021594    Male      5
3  tt0026274    None      0
4  tt0026274  Female     11
5  tt0026274    Male      9
6  tt0027256    None      0
7  tt0027256  Female      5
8  tt0027256    Male      8
9  tt0028217  Female      3
```

```
True
```

```
         MID  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 189 ms, sys: 7.96 ms, total: 197 ms
Wall time: 197 ms
```

[22]:
```
query5a = """
WITH
    q5aa AS ({}),
    q5ab AS ({})
```

```
SELECT
    CAST(SUBSTRING(M.year, -4) AS INTEGER) YEAR,
    COUNT(M.title) Female_Cast_Only_Movies
FROM
    Movie AS M
JOIN
    q5aa
ON
    TRIM(M.MID) = TRIM(q5aa.MID)
WHERE
    TRIM(M.MID) NOT IN (
        SELECT
            TRIM(q5ab.MID)
        FROM
            q5ab
    )
GROUP BY
    YEAR
""".format(q_5aa, q_5ab)
```

[23]:
```
%%time

def grader_5a(q5a):
    q5a_results  = pd.read_sql_query(sql=q5a, con=conn)
    display(q5a_results.head(n=10))
    assert (q5a_results.shape == (4, 2))

grader_5a(q5a=query5a)
```

```
   YEAR  Female_Cast_Only_Movies
0  1939                        1
1  1999                        1
2  2000                        1
3  2018                        1

CPU times: user 194 ms, sys: 2.98 ms, total: 197 ms
Wall time: 196 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.

[24]:
```
query5b = """
WITH
    q5a AS ({})
SELECT
    CAST(SUBSTRING(M.year, -4) AS INTEGER) YEAR,
    (q5a.Female_Cast_Only_Movies * 1.0) / COUNT(*) Percentage_Female_Only_Movie,
```

```
    COUNT(M.title) Total_Movies
FROM
    Movie AS M
JOIN
    q5a
ON
    CAST(SUBSTRING(q5a.year, -4) AS INTEGER) = CAST(SUBSTRING(M.year, -4) AS␣
 ↪INTEGER)
GROUP BY
    CAST(SUBSTRING(M.year, -4) AS INTEGER)
""".format(query5a)
```

[25]:
```
%%time

def grader_5b(q5b):
    q5b_results  = pd.read_sql_query(sql=q5b, con=conn)
    display(q5b_results.head(n=10))
    assert (q5b_results.shape == (4, 3))

grader_5b(q5b=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 191 ms, sys: 7.98 ms, total: 199 ms
Wall time: 199 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.

[26]:
```
q_6a = """
SELECT
    MC.MID,
    COUNT(DISTINCT TRIM(MC.PID)) count
FROM
    M_Cast AS MC
GROUP BY
    MC.MID
"""
```

[27]:
```
query6 = """
WITH
    q6a AS ({})
```

```
SELECT
    TRIM(M.title) title,
    q6a.count
FROM
    Movie AS M
JOIN
    q6a
ON
    TRIM(M.MID) = TRIM(q6a.MID)
ORDER BY
    q6a.count DESC
""".format(q_6a)
```

[28]:
```
%%time

def grader_6(q6):
    q6_results  = pd.read_sql_query(sql=q6, con=conn)
    display(q6_results.head(n=10))
    assert (q6_results.shape == (3473, 2))

grader_6(q6=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 834 ms, sys: 2.91 ms, total: 837 ms
Wall time: 838 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

a) Write a query that computes number of movies in each year.

[29]:
```
query7a = """
SELECT
    CAST(SUBSTRING(M.year, -4) AS INTEGER) Movie_Year,
```

```
    COUNT(TRIM(M.title)) Total_Movies
FROM
    Movie AS M
GROUP BY
    Movie_Year
"""
```

[30]:
```
%%time

def grader_7a(q7a):
    q7a_results  = pd.read_sql_query(sql=q7a, con=conn)
    display(q7a_results.head(n=10))
    assert (q7a_results.shape == (78, 2))

grader_7a(q7a=query7a)
```

	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 3.51 ms, sys: 970 µs, total: 4.48 ms
Wall time: 3.83 ms
```

b) 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 more than or equal current_year

[31]:
```
query7b = """
WITH
    q7a1 AS ({}),
    q7a2 AS ({})
SELECT
    q7a1.Movie_Year,
    q7a1.Total_Movies,
    q7a2.Movie_Year,
    q7a2.Total_Movies
FROM
    q7a1
JOIN
    q7a2
ON
```

```
    q7a2.Movie_Year <= q7a1.Movie_Year + 9
AND
    q7a2.Movie_Year >= q7a1.Movie_Year
""".format(query7a, query7a)
```

[32]:
```
%%time

def grader_7b(q7b):
    q7b_results  = pd.read_sql_query(sql=q7b, con=conn)
    display(q7b_results.head(n=10))
    assert (q7b_results.shape == (713, 4))

grader_7b(q7b=query7b)
```

	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 7.09 ms, sys: 976 µs, total: 8.07 ms
Wall time: 7.32 ms
```

Write a query that will return the decade that has maximum number of movies.

[33]:
```
query7 = """
WITH
    q7a1 AS ({}),
    q7a2 AS ({}),
    q7 AS (
        SELECT
            q7a1.Movie_Year,
            SUM(q7a2.Total_Movies) Total_Movies
        FROM
            q7a1
        JOIN
            q7a2
        ON
            q7a2.Movie_Year <= q7a1.Movie_Year + 9
        AND
            q7a2.Movie_Year >= q7a1.Movie_Year
        GROUP BY
            q7a1.Movie_Year
```

```
    )
SELECT
    q7.Movie_Year Decade,
    MAX(q7.Total_Movies) Decade_Movie_Count
FROM
    q7
""".format(query7a, query7a)
```

[34]:
```
%%time

def grader_7(q7):
    q7_results  = pd.read_sql_query(sql=q7, con=conn)
    display(q7_results.head(n=10))
    assert (q7_results.shape == (1, 2))

grader_7(q7=query7)
```

```
   Decade  Decade_Movie_Count
0    2008                1203

CPU times: user 6.16 ms, sys: 994 µs, total: 7.15 ms
Wall time: 6.31 ms
```

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

a) Write a query that will results in number of movies actor-director worked together.

[35]:
```
query8a = """
SELECT
    TRIM(MC.PID) actor,
    TRIM(MD.PID) director,
    COUNT(TRIM(M.title)) movies
FROM
    M_Cast AS MC
JOIN
    M_Director AS MD
ON
    TRIM(MC.MID) = TRIM(MD.MID)
JOIN
    Movie AS M
ON
    TRIM(MD.MID) = TRIM(M.MID)
GROUP BY
    actor,
    director
"""
```

[36]:
```
%%time

def grader_8a(q8a):
    q8a_results  = pd.read_sql_query(sql=q8a, con=conn)
    display(q8a_results.head(n=10))
    assert (q8a_results.shape == (73408, 3))

grader_8a(q8a=query8a)
```

```
      actor   director  movies
0  nm0000002  nm0496746       1
1  nm0000027  nm0000180       1
2  nm0000039  nm0896533       1
3  nm0000042  nm0896533       1
4  nm0000047  nm0004292       1
5  nm0000073  nm0485943       1
6  nm0000076  nm0000229       1
7  nm0000092  nm0178997       1
8  nm0000093  nm0000269       1
9  nm0000096  nm0113819       1
```

```
CPU times: user 40.7 s, sys: 13.5 ms, total: 40.8 s
Wall time: 40.8 s
```

Some research on problem 8.

[37]:
```
def query_df_info(query):
    """
    This funtion gives useful information about the data based on query.
    """
    df = pd.read_sql_query(sql=query, con=conn)
    display(df.head(n=10))
    print(df.shape)
    return None
```

All distinct directors.

[38]:
```
%%time

query_directors = """
SELECT
    DISTINCT
        TRIM(P.PID) DID,
        TRIM(P.Name) Name
FROM
    M_Director AS MD
JOIN
    Person AS P
ON
```

```
    TRIM(MD.PID) = TRIM(P.PID)
"""

query_df_info(query=query_directors)
```

```
         DID             Name
0  nm0785227      Andy Serkis
1  nm0002657        Gary Ross
2  nm1012385      Roar Uthaug
3  nm0923736      Joss Whedon
4  nm9751348  Rahi Anil Barve
5  nm0438461  Abhishek Kapoor
6  nm0751577    Anthony Russo
7  nm1437189  Sriram Raghavan
8  nm0204628      Garth Davis
9  nm1636742      Leena Yadav

(1462, 2)
CPU times: user 9.7 s, sys: 3.94 ms, total: 9.71 s
Wall time: 9.73 s
```

Directors other than Yash Chopra.

[39]:
```
%%time

query_nyc_pid = """
WITH
    AllD AS (
        SELECT
            DISTINCT
                TRIM(P.PID) DID,
                TRIM(P.Name) Name
        FROM
            M_Director AS MD
        JOIN
            Person AS P
        ON
            TRIM(MD.PID) = TRIM(P.PID)
    )
SELECT
    TRIM(P.PID) PID
FROM
    Person AS P
JOIN
    AllD
ON
    TRIM(P.PID) = TRIM(AllD.DID)
WHERE
```

```
    TRIM(P.Name) <> "Yash Chopra"
"""

query_df_info(query=query_nyc_pid)
```

```
         PID
0  nm0785227
1  nm0001162
2  nm0438501
3  nm0795661
4  nm0542498
5  nm2945270
6  nm2147526
7  nm0000187
8  nm0704694
9  nm0451148

(1461, 1)
CPU times: user 13.5 s, sys: 949 µs, total: 13.5 s
Wall time: 13.6 s
```

Movies not directed by Yash Chopra but by other directors.

[40]:
```
%%time

query_nyc_mid = """
WITH
    AllD AS (
        SELECT
            DISTINCT
                TRIM(P.PID) DID,
                TRIM(P.Name) Name
        FROM
            M_Director AS MD
        JOIN
            Person AS P
        ON
            TRIM(MD.PID) = TRIM(P.PID)
    ),
    NYCPID AS (
        SELECT
            TRIM(P.PID) PID
        FROM
            Person AS P
        JOIN
            AllD
        ON
            TRIM(P.PID) = TRIM(AllD.DID)
```

```
        WHERE
            TRIM(P.Name) <> "Yash Chopra"
    )
SELECT
    TRIM(MD.MID) MID
FROM
    M_Director AS MD
JOIN
    NYCPID
ON
    TRIM(MD.PID) = TRIM(NYCPID.PID)
"""

query_df_info(query=query_nyc_mid)
```

```
         MID
0  tt2388771
1  tt0809504
2  tt0149568
3  tt1340778
4  tt1772332
5  tt0449870
6  tt1210356
7  tt0220832
8  tt0255309
9  tt0263491
```

```
(3452, 1)
CPU times: user 33.4 s, sys: 0 ns, total: 33.4 s
Wall time: 33.6 s
```

Yash Chopra's details - ID, movies and total cast for each movie.

[41]:
```
%%time

query_yc_pid = """
SELECT
    TRIM(P.PID) PID,
    TRIM(P.Name) Name
FROM
    Person AS P
WHERE
    TRIM(P.Name) = "Yash Chopra"
"""

print("Yash Chopra's ID")
query_df_info(query=query_yc_pid)
```

```
Yash Chopra's ID
```

```
         PID         Name
0  nm0007181  Yash Chopra
```

```
(1, 2)
CPU times: user 7.13 ms, sys: 7 µs, total: 7.14 ms
Wall time: 5.78 ms
```

[42]:
```
%%time

query_yc_mids = """
WITH
    YCPID AS (
        SELECT
            TRIM(P.PID) PID,
            TRIM(P.Name) Name
        FROM
            Person AS P
        WHERE
            TRIM(P.Name) = "Yash Chopra"
    )
SELECT
    TRIM(M.MID) MID,
    TRIM(M.title) Movie,
    CAST(SUBSTRING(M.year, -4) AS INTEGER) Year
FROM
    Movie AS M
JOIN
    M_Director AS MD
ON
    TRIM(M.MID) = TRIM(MD.MID)
JOIN
    YCPID
ON
    TRIM(MD.PID) = TRIM(YCPID.PID)
ORDER BY
    Year DESC
"""

print("Yash Chopra's movies")
query_df_info(query=query_yc_mids)
```

```
Yash Chopra's movies
```

	MID	Movie	Year
0	tt2176013	Jab Tak Hai Jaan	2012
1	tt0420332	Veer-Zaara	2004
2	tt0118983	Dil To Pagal Hai	1997
3	tt0109555	Darr	1993
4	tt0107777	Parampara	1993

```
5  tt0102258             Lamhe  1991
6  tt0097045           Chandni  1989
7  tt0096390             Vijay  1988
8  tt0085912           Mashaal  1984
9  tt0083081           Silsila  1981
```

```
(21, 3)
CPU times: user 847 ms, sys: 0 ns, total: 847 ms
Wall time: 848 ms
```

[43]:
```
%%time

query_yc_movie_cast_size = """
WITH
    YCPID AS (
        SELECT
            TRIM(P.PID) PID,
            TRIM(P.Name) Name
        FROM
            Person AS P
        WHERE
            TRIM(P.Name) = "Yash Chopra"
    ),
    YCM AS (
        SELECT
            TRIM(M.MID) MID,
            TRIM(M.title) Movie
        FROM
            Movie AS M
        JOIN
            M_Director AS MD
        ON
            TRIM(M.MID) = TRIM(MD.MID)
        JOIN
            YCPID
        ON
            TRIM(MD.PID) = TRIM(YCPID.PID)
    )
SELECT
    TRIM(YCM.MID) MID,
    COUNT(DISTINCT TRIM(MC.PID)) Cast_Size
FROM
    M_Cast AS MC
JOIN
    YCM
ON
    TRIM(MC.MID) = TRIM(YCM.MID)
GROUP BY
```

```
    TRIM(YCM.MID)
"""

print("Yash Chopra's cast size for each movie")
query_df_info(query=query_yc_movie_cast_size)
```

```
Yash Chopra's cast size for each movie

         MID  Cast_Size
0  tt0052736         24
1  tt0059893         30
2  tt0064506         12
3  tt0072860         36
4  tt0074730         11
5  tt0078418         18
6  tt0079386         53
7  tt0083081         24
8  tt0085912         17
9  tt0096390         21

(21, 2)
CPU times: user 1min 12s, sys: 5.82 ms, total: 1min 12s
Wall time: 1min 12s
```

Actors who acted in Yash Chopra's movies.

[44]:
```
%%time

query_yc_cast = """
WITH
    YCPID AS (
        SELECT
            TRIM(P.PID) PID,
            TRIM(P.Name) Name
        FROM
            Person AS P
        WHERE
            TRIM(P.Name) = "Yash Chopra"
    ),
    YCM AS (
        SELECT
            TRIM(M.MID) MID,
            TRIM(M.title) Movie
        FROM
            Movie AS M
        JOIN
            M_Director AS MD
        ON
            TRIM(M.MID) = TRIM(MD.MID)
```

```
        JOIN
            YCPID
        ON
            TRIM(MD.PID) = TRIM(YCPID.PID)
    )
SELECT
    TRIM(MC.PID) CID,
    TRIM(P.Name) Actor,
    YCM.MID,
    YCM.Movie
FROM
    YCM
JOIN
    M_Cast AS MC
ON
    TRIM(YCM.MID) = TRIM(MC.MID)
JOIN
    Person AS P
ON
    TRIM(MC.PID) = TRIM(P.PID)
"""

print("Cast who acted in Yash Chopra's movies")
query_df_info(query=query_yc_cast)
```

```
Cast who acted in Yash Chopra's movies

         CID              Actor        MID       Movie
0  nm0451321    Shah Rukh Khan  tt0420332  Veer-Zaara
1  nm0611552      Rani Mukerji  tt0420332  Veer-Zaara
2  nm0006689      Preity Zinta  tt0420332  Veer-Zaara
3  nm0451601        Kiron Kher  tt0420332  Veer-Zaara
4  nm0244890       Divya Dutta  tt0420332  Veer-Zaara
5  nm1224082       Boman Irani  tt0420332  Veer-Zaara
6  nm0451600       Anupam Kher  tt0420332  Veer-Zaara
7  nm0000821  Amitabh Bachchan  tt0420332  Veer-Zaara
8  nm0004564       Hema Malini  tt0420332  Veer-Zaara
9  nm0048075    Manoj Bajpayee  tt0420332  Veer-Zaara

(589, 4)
CPU times: user 2.96 s, sys: 9.98 ms, total: 2.97 s
Wall time: 2.97 s
```

From the above cast details, how many times each actor was casted in Yash Chopra's movies?

[45]:
```
%%time

query_cast_how_many_times_yc = """
WITH
```

```
    YCPID AS (
        SELECT
            TRIM(P.PID) PID,
            TRIM(P.Name) Name
        FROM
            Person AS P
        WHERE
            TRIM(P.Name) = "Yash Chopra"
    ),
    YCM AS (
        SELECT
            TRIM(M.MID) MID,
            TRIM(M.title) Movie
        FROM
            Movie AS M
        JOIN
            M_Director AS MD
        ON
            TRIM(M.MID) = TRIM(MD.MID)
        JOIN
            YCPID
        ON
            TRIM(MD.PID) = TRIM(YCPID.PID)
    ),
    CYC AS (
        SELECT
            TRIM(MC.PID) CID,
            TRIM(P.Name) Actor,
            YCM.MID,
            YCM.Movie
        FROM
            YCM
        JOIN
            M_Cast AS MC
        ON
            TRIM(YCM.MID) = TRIM(MC.MID)
        JOIN
            Person AS P
        ON
            TRIM(MC.PID) = TRIM(P.PID)
    )
SELECT
    CYC.CID,
    CYC.Actor,
    COUNT(CYC.CID) YC_Movies
FROM
    CYC
```

```
GROUP BY
    CYC.CID
"""

print("How many times each actor casted in Yash Chopra's movies?")
query_df_info(query=query_cast_how_many_times_yc)
```

How many times each actor casted in Yash Chopra's movies?

	CID	Actor	YC_Movies
0	nm0000821	Amitabh Bachchan	6
1	nm0002043	Madhuri Dixit	1
2	nm0004109	Gulshan Grover	2
3	nm0004334	Rekha	1
4	nm0004429	Dharmendra	1
5	nm0004434	Shashi Kapoor	7
6	nm0004435	Rajesh Khanna	3
7	nm0004437	Sridevi	2
8	nm0004487	Juhi Chawla	2
9	nm0004564	Hema Malini	4

(430, 3)
CPU times: user 3.05 s, sys: 13.9 ms, total: 3.07 s
Wall time: 3.08 s

[46]:
```
query8 = """
SELECT (
    SELECT
        Name
    FROM
        Person
    WHERE
        TRIM(PID) = actor
    )
    Name,
    movies
FROM (
    {}
)
WHERE
    (actor, movies) IN (
        SELECT
            actor,
            MAX(movies)
        FROM (
            {}
        )
        GROUP BY
```

```
            actor
    )
AND
    director = (
        SELECT
            TRIM(PID)
        FROM
            Person
        WHERE
            TRIM(Name) = "Yash Chopra"
    )
ORDER BY
    movies DESC
""".format(query8a, query8a)
```

[47]:
```
%%time

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

grader_8(q8=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 1min 20s, sys: 15.9 ms, total: 1min 20s
Wall time: 1min 20s
```

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, and so on.

Return all actors whose Shahrukh number is 2.

[48]:
```
%%time

query_srk = """
SELECT
    *
FROM
    Person
WHERE
    TRIM(Name) = "Shah Rukh Khan"
"""

query_df_info(query=query_srk)
```

```
   index        PID              Name Gender
0   3012  nm0451321   Shah Rukh Khan   Male
```

```
(1, 4)
CPU times: user 8.5 ms, sys: 1.98 ms, total: 10.5 ms
Wall time: 9.52 ms
```

[49]:
```
query9a = """
WITH
    SO AS (
        SELECT
            TRIM(P.PID) SRKPID
        FROM
            Person AS P
        WHERE
            TRIM(P.Name) = "Shah Rukh Khan"
    ),
    SRKM AS (
        SELECT
            MC.MID SRKMID,
            SO.SRKPID SRKPID
        FROM
            M_Cast AS MC
        JOIN
            SO
        ON
            TRIM(MC.PID) = TRIM(SO.SRKPID)
    )
SELECT
    DISTINCT
        TRIM(MC.PID) S1_PID
```

```
FROM
    M_Cast AS MC
JOIN
    SRKM
ON
    TRIM(MC.MID) = SRKM.SRKMID
AND
    TRIM(MC.PID) <> SRKM.SRKPID
"""
```

[50]:
```
%%time

def grader_9a(q9a):
    q9a_results  = pd.read_sql_query(sql=q9a, con=conn)
    display(q9a_results.head(n=10))
    print(q9a_results.shape)
    assert (q9a_results.shape == (2382, 1))

grader_9a(q9a=query9a)
```

```
     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 745 ms, sys: 2.99 ms, total: 748 ms
Wall time: 749 ms
```

[51]:
```
query9 = """
WITH
    SO AS (
        SELECT
            TRIM(P.PID) SRKPID
        FROM
            Person AS P
        WHERE
            TRIM(P.Name) = "Shah Rukh Khan"
    ),
    SRKM AS (
        SELECT
```

```
            DISTINCT
                MC.MID SRKMID,
                S0.SRKPID SRKPID
        FROM
            M_Cast AS MC
        JOIN
            S0
        ON
            TRIM(MC.PID) = S0.SRKPID
    ),
    S1C AS (
        SELECT
            DISTINCT
                TRIM(MC.PID) S1_PID
        FROM
            M_Cast AS MC
        JOIN
            SRKM
        ON
            TRIM(MC.MID) = SRKM.SRKMID
        AND
            TRIM(MC.PID) <> SRKM.SRKPID
    ),
    S1M AS (
        SELECT
            DISTINCT
                MC.MID S1MID,
                S1C.S1_PID S1_PID
        FROM
            M_Cast AS MC
        JOIN
            S1C
        ON
            TRIM(MC.PID) = S1C.S1_PID
    ),
    S2C AS (
        SELECT
            DISTINCT
                TRIM(MC.PID) S2_PID
        FROM
            M_Cast AS MC
        JOIN
            S1M
        ON
            TRIM(MC.MID) = S1M.S1MID
        AND
            TRIM(MC.PID) <> S1M.S1_PID
```

```
    )
SELECT
    DISTINCT
        TRIM(P.Name) Actor_Name
FROM
    Person AS P
JOIN
    S2C
ON
    P.PID = S2C.S2_PID
"""
```

[52]:
```
%%time

def grader_9(q9):
    q9_results  = pd.read_sql_query(sql=q9, con=conn)
    display(q9_results.head(n=10))
    print(q9_results.shape)
    assert (q9_results.shape == (25698, 1))

# grader_9(q9=query9)
query_df_info(query=query9)
```

```
             Actor_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

(26521, 1)
CPU times: user 555 ms, sys: 5.97 ms, total: 561 ms
Wall time: 562 ms
```

End of the file.