sql

June 5, 2022

1 SQL Assignment

[2]:
```
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 this link
# https://drive.google.com/file/d/1O-1-L1DdNxEK6O6nG2jS31MbrMh-OnXM/view?
 ↪usp=sharing
```

[3]:
```
conn = sqlite3.connect("Db-IMDB-Assignment.db")
```

Overview of all tables

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


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 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(*)

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.

[116]:
```
%%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 director, title movie, year from
(
    select d.pid as id, m.title, m.year from movie m inner join m_director d on␣
 ↪m.mid=d.mid where m.mid in (
        select mid from m_genre where gid in(
            select gid from genre where trim(name) like '%Comedy%'
        )
    ) and  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)
) inner join person p on id=p.pid
"""
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
Wall time: 142 ms
```

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

[117]:
```
%%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 Actor from Person P inner join
    M_Cast MC on P.PID = trim(MC.PID) inner join
        Movie M on MC.MID = M.MID where trim(M.title) = 'Anand'
"""
grader_2(query2)
```

```
              Actor
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
Wall time: 361 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.)

[109]:
```
%%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
Wall time: 520 ms
```

[7]:
```
%%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 Actor from person where pid in
(
    select distinct trim(MC.PID) from M_Cast MC inner join
        Movie M on MC.MID = M.MID WHERE CAST(SUBSTR(M.year,-4) as Integer)<1970
    intersect
    select distinct trim(MC.PID) from M_Cast MC inner join
        Movie M on MC.MID = M.MID WHERE CAST(SUBSTR(M.year,-4) as Integer)>1990
)
"""
grader_3(query3)
```

```
              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
Wall time: 608 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.

[7]:
```
%%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 Director_ID, count(*) 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
Wall time: 12 ms
```

[118]:
```
%%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 Director, no.movies movies from
(
    select PID, count(*) movies from M_Director group by PID having movies >= 10
) no inner join person p on no.PID = p.PID order by movies desc
"""
grader_4(query4)
```

```
               Director  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
Wall time: 35 ms
```

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

[19]:
```
%%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 mc.mid, p.gender, count(*) Count from M_Cast mc inner join
    Person p on trim(mc.pid)=p.pid group by mid, gender
"""

print(grader_5aa(query_5aa))

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

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

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
Wall time: 691 ms
```

[8]:
```
%%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 count(*) no_movies, CAST(SUBSTR(year,-4) as UNSIGNED) year from Movie
WHERE MID IN (
    select mc.mid mid from M_Cast mc inner join
        Person p on trim(mc.pid)=p.pid where p.gender='Female' group by mid
```

```
    except
        select mc.mid mid from M_Cast mc inner join
            Person p on trim(mc.pid)=p.pid where p.gender='Male' group by mid
) GROUP BY year
"""
grader_5a(query5a)
```

```
   no_movies  year
0          1  1939
1          1  1999
2          1  2000
3          1  2018
Wall time: 495 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.

[121]:
```
%%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 m.year, cast(fe_o.no_movies as real)/count(*) percentage,
count(*) total_movies from movie m inner join (
    SELECT count(*) no_movies, CAST(SUBSTR(year,-4) as UNSIGNED) year from Movie
    WHERE MID IN (
            select mc.mid mid from M_Cast mc inner join
                Person p on trim(mc.pid)=p.pid where p.gender='Female' group by␣
 ↪mid
            except
                select mc.mid mid from M_Cast mc inner join
                    Person p on trim(mc.pid)=p.pid where p.gender='Male' group␣
 ↪by mid
    ) GROUP BY year
) fe_o on CAST(SUBSTR(m.year,-4) as UNSIGNED)=fe_o.year group by CAST(SUBSTR(m.
 ↪year,-4) as UNSIGNED)
"""
grader_5b(query5b)
```

```
   year  percentage  total_movies
0  1939    0.500000             2
1  1999    0.015152            66
```

```
2  2000    0.015625            64
3  2018    0.009615           104
Wall time: 517 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.

[123]:
```
%%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 fe.cast_size, m.title from Movie m inner join (
select count(*) cast_size, MID mid from M_Cast group by MID
) fe on m.MID=fe.mid order by fe.cast_size desc
"""
grader_6(query6)
```

```
   cast_size                        title
0        238                Ocean's Eight
1        233                     Apaharan
2        215                         Gold
3        213             My Name Is Khan
4        191  Captain America: Civil War
5        170                     Geostorm
6        165                      Striker
7        154                         2012
8        144                       Pixels
9        140       Yamla Pagla Deewana 2
Wall time: 112 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

[82]:
```
%%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 cast(substr(year,-4) as int) Movie_Year, count(*) Total_Movies from␣
 ↪movie group by Movie_Year
"""
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
Wall time: 11 ms
```

[85]:
```
%%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 y.y Movie_Year, y.t Total_Movies, m.y Movie_Year, m.t Total_Movies  from
(select cast(substr(year,-4) as int) y, count(*) t from movie group by y) y␣
 ↪inner join
```

```
(select cast(substr(year,-4) as int) y, count(*) t from movie group by y) m on
m.y >= y.y and m.y < y.y + 10
"""
grader_7b(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
Wall time: 21 ms
```

[108]:
```
%%time
def grader_7(q7):
    q7_results  = pd.read_sql_query(q7,conn)
    print(q7_results.head(10))
    assert (q7_results.shape == (1, 2))

# select y.year decade_start, count(*) tot_movies from
# (select distinct cast(substr(year,-4) as int) year from movie) y inner join
# (select cast(substr(year,-4) as int) year from movie) m on m.year >= y.year␣
 ↪and m.year < y.year + 10
# group by y.year order by count(*) desc limit 1

query7 = """
select y.y decade_start, sum(m.t) Total_Movies from
(select cast(substr(year,-4) as int) y, count(*) t from movie group by y) y␣
 ↪inner join
(select cast(substr(year,-4) as int) y, count(*) t from movie group by y) m on
m.y >= y.y and m.y < y.y + 10 group by decade_start order by Total_Movies desc␣
 ↪limit 1
"""
grader_7(query7)
```

```
   decade_start  Total_Movies
0          2008          1203
Wall time: 17 ms
```

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

[124]:
```
%%time
def grader_8a(q8a):
    q8a_results  = pd.read_sql_query(q8a,conn)
    print(q8a_results.head(10))
    assert (q8a_results.shape == (73408, 3))

query8a = """
select trim(mc.pid) actor, md.pid director, count(*) movies
from m_director md inner join
    m_cast mc on md.mid=mc.mid
        group by actor,director
"""
grader_8a(query8a)

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

```
      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
Wall time: 780 ms
```

[127]:
```
%%time

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

query8 = """
select p.name, movies from
(
    select trim(mc.pid) a, md.pid d, count(*) movies, rank() over (partition by␣
 ↪mc.pid order by count(*) desc) rn
    from m_director md inner join
        m_cast mc on md.mid=mc.mid
```

```
            group by a,d
) inner join
    person p on a=trim(p.pid)
    where rn =1 and d like (select pid from person where trim(name) = 'Yash␣
 ↪Chopra') 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     Waheeda Rehman       5
5      Rakhee Gulzar       5
6     Achala Sachdev       4
7        Neetu Singh       4
8           Ravikant       4
9    Parikshat Sahni       3
(245, 2)
Wall time: 1.18 s
```

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.

[33]:
```
%%time
def grader_9a(q9a):
    q9a_results  = pd.read_sql_query(q9a,conn)
    print(q9a_results.head(10))
    print(q9a_results.shape)
    assert (q9a_results.shape == (2382, 1))

query9a = """
select distinct pid S1_PID from m_cast where mid in
(
    select mid from m_cast where trim(pid) = (select pid from person where␣
 ↪trim(name) = 'Shah Rukh Khan')
) and trim(pid) != (select pid from person where trim(name) = '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 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)
Wall time: 98.9 ms
```

[34]:
```
%%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 person where pid in
(
    select trim(pid) from m_cast where mid in
    (
        select mid from m_cast where pid in
        (
            select distinct pid from m_cast where mid in
            (
                select mid from m_cast where trim(pid) = (select pid from␣
 ↪person where trim(name) = 'Shah Rukh Khan')
            ) and trim(pid) != (select pid from person where trim(name) = 'Shah␣
 ↪Rukh Khan')
        )
    ) and pid not in
    (
            select distinct pid from m_cast where mid in
            (
                select mid from m_cast where trim(pid) = (select pid from␣
 ↪person where trim(name) = '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)
Wall time: 622 ms
```

[ ]: