parth.pandey13103447@gmail.com_18

October 4, 2021

1 SQL Assignment

1.1 Imports

[1]:
```
import pandas as pd
import sqlite3, os
from IPython.display import display, HTML
```

[2]:
```
os.getcwd()
```

[2]:
```
'/home/parth/AppliedAI/assignments/18 SQL Assignment on IMDB data'
```

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

1.2 Overview of all tables

[4]:
```
tables = pd.read_sql_query("SELECT NAME AS 'Table_Name' FROM sqlite_master␣
 ↪WHERE type='table'",conn)
tables = tables["Table_Name"].values.tolist()
```

[5]:
```
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
--------------------------------------------------------------------------------
--------------------
```

1.3 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(*)

1.4 Questions

1.4.1 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.

Hint

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.

[6]:
```
%%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 = """
with  genre_table AS (
  SELECT
    mid
  FROM M_GENRE mg
  JOIN (
    SELECT
      gid
    FROM GENRE
    WHERE LOWER(name) LIKE '%comedy%'
  ) g
  ON g.gid = mg.gid
),
movie_table AS (
  SELECT
    m.mid,
    TRIM(title) movie_name,
    CAST(SUBSTR(TRIM(year),-4) AS INTEGER) as year
  FROM MOVIE m
  JOIN genre_table gt
  ON gt.mid = m.mid
  WHERE (
  CAST(SUBSTR(TRIM(year),-4) AS INTEGER) % 4 = 0 OR (CAST(SUBSTR(TRIM(year),-4)␣
 ↪AS INTEGER) % 100 = 0
```

```
  OR CAST(SUBSTR(TRIM(year),-4) AS INTEGER) % 400 = 0
)
      )
)
SELECT
  UPPER(p.name) AS Name,
  UPPER(mt.movie_name) AS Movie_Name,
  mt.year as Year
FROM movie_table mt
JOIN M_Director md
ON md.mid = mt.mid
JOIN PERSON p
ON p.pid=md.pid
"""
grader_1(query1)
```

```
                Name                          Movie_Name  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 112 ms, sys: 426 µs, total: 113 ms
Wall time: 112 ms
```

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

[7]:
```
%%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
    UPPER(p.name)
FROM MOVIE m
JOIN M_CAST mc
ON m.mid = mc.mid
JOIN PERSON p
ON TRIM(mc.pid)  =TRIM(p.pid)
```

```
WHERE LOWER(TRIM(m.title)) = 'anand'
  AND CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER) = 1971

"""
grader_2(query2)
```

```
       UPPER(p.name)
0   AMITABH BACHCHAN
1      RAJESH KHANNA
2      SUMITA SANYAL
3         RAMESH DEO
4          SEEMA DEO
5     ASIT KUMAR SEN
6         DEV KISHAN
7       ATAM PRAKASH
8      LALITA KUMARI
9             SAVITA
CPU times: user 288 ms, sys: 0 ns, total: 288 ms
Wall time: 286 ms
```

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

[8]:
```
%%time

def grader_3a(query_less_1970, query_more_1990):
    q3_a = pd.read_sql_query(query_less_1970,conn)
    q3_b = pd.read_sql_query(query_more_1990,conn)
    print(q3_a.shape, q3_b.shape)
    return (q3_a.shape == (4942,1)) and (q3_b.shape == (62570,1))

query_less_1970 ="""
SELECT
    LOWER(TRIM(mc.pid)) pid
FROM M_CAST mc
JOIN MOVIE m ON m.mid = mc.mid
WHERE CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER) < 1970
 """
query_more_1990 =""" SELECT
     LOWER(TRIM(mc.pid)) pid
FROM MOVIE m
JOIN M_CAST mc ON m.mid = mc.mid

WHERE CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER) > 1990
 """
print(grader_3a(query_less_1970, query_more_1990))
```

```
# using the above two queries, you can find the answer to the given question
```

```
(4942, 1) (62570, 1)
True
CPU times: user 257 ms, sys: 7.78 ms, total: 265 ms
Wall time: 263 ms
```

[9]:
```
%%time
def grader_3(q3):
    q3_results  = pd.read_sql_query(q3,conn)
    print(q3_results.head(10))
    print(q3_results.shape)
    assert (q3_results.shape == (300,1))

query3 ="""
with temp as (
    SELECT
       distinct TRIM(mc.pid) pid, CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER) year
    FROM M_CAST mc
    JOIN MOVIE m ON trim(m.mid) = trim(mc.mid)
    )
select name
from person
where trim(pid) in (
select
distinct a.pid
from (select pid from temp t where year < 1970 ) a
join ( select pid from temp where year > 1990) b on a.pid = b.pid
)
"""
grader_3(query3)
```

```
                Name
0       Rishi Kapoor
1   Amitabh Bachchan
2             Asrani
3       Zohra Sehgal
4    Parikshat Sahni
5      Rakesh Sharma
6        Sanjay Dutt
7          Ric Young
8              Yusuf
9     Suhasini Mulay
(300, 1)
CPU times: user 48.4 s, sys: 5.36 s, total: 53.8 s
Wall time: 53.8 s
```

1.4.4 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.

[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 p.name Director_Name , count(md.mid) Movie_Count
FROM person p
JOIN m_director md
ON p.pid = md.pid
GROUP BY Director_Name
HAVING Movie_Count >= 10
ORDER BY Movie_Count DESC
 """
grader_4(query4)
```

```
           Director_Name  Movie_Count
0            David Dhawan           39
1            Mahesh Bhatt           36
2            Priyadarshan           30
3         Ram Gopal Varma           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 64.9 ms, sys: 981 µs, total: 65.9 ms
Wall time: 73.7 ms
```

1.4.5 Q5

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

[11]:
```
%%time
def grader_5a(q5a):
    q5a_results  = pd.read_sql_query(q5a,conn)
    print(q5a_results.head(100))
    print(q5a_results.shape)
    assert (q5a_results.shape == (4,2))

query5a ="""
select movie_year, count(distinct mid ) num_movies
```

```
from (
    select m.mid, m.year movie_year, max(p.gender = 'Male') has_male_actor
    from movie m
    inner join m_cast mc on mc.mid = m.mid
    inner join person p on p.pid = trim(mc.pid)
    group by m.mid, m.year
) t
where has_male_actor = 0
group by movie_year
"""
grader_5a(query5a)
```

```
  movie_year  num_movies
0       1939           1
1       1999           1
2       2000           1
3     I 2018           1
(4, 2)
CPU times: user 390 ms, sys: 27.4 ms, total: 418 ms
Wall time: 418 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.

[12]:
```
%%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 = """
with temp as (
     select m.mid, m.year movie_year, max(p.gender = 'Male') has_male_actor
    from movie m
    inner join m_cast mc on mc.mid = m.mid
    inner join person p on p.pid = trim(mc.pid)
    group by m.mid, m.year
),
temp2 as (
select movie_year, count(distinct mid) no_movies
from temp t
where has_male_actor = 0
group by movie_year),
temp3 as (
select movie_year, count( distinct mid ) tot_movies
```

```
from temp t
group by movie_year
)
select t2.movie_year, t2.no_movies, round(t2.no_movies/t3.tot_movies,5) as ratio
from temp2 t2
join temp3 t3 on t2.movie_year = t3.movie_year

 """
grader_5b(query5b)
```

```
  movie_year  no_movies  ratio
0       1939          1    0.0
1       1999          1    0.0
2       2000          1    0.0
3     I 2018          1    0.0
CPU times: user 678 ms, sys: 16.2 ms, total: 694 ms
Wall time: 693 ms
```

1.4.6 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.

[13]:
```
%%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 m.title, a.cast_size
FROM
(
SELECT mid, count ( distinct (pid )) cast_size
FROM m_cast
group by 1
) a
left join movie m ON a.mid = m.mid
order by a.cast_size desc
 """
grader_6(query6)
```

```
                      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 80.6 ms, sys: 12.1 ms, total: 92.7 ms
Wall time: 91.6 ms
```

1.4.7 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.

[14]:
```
%%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 start || "-"  || end grp, count(*) count
from (
select CAST(SUBSTR(TRIM(year),-4) AS STRING) start,␣
 ↪CAST(CAST(SUBSTR(TRIM(year),-4) AS INTEGER) + 9 AS STRING) end ,␣
 ↪CAST(SUBSTR(TRIM(year),-4) AS INTEGER) year

from movie
)
group by 1
order by count desc
limit 1
"""
grader_7(query7)
# if you check the output we are printinng all the year in that decade, its␣
 ↪fine you can print 2008 or 2008-2017
```

```
         grp  count
0  2013-2022    136
CPU times: user 10.7 ms, sys: 182 µs, total: 10.9 ms
Wall time: 9.33 ms
```

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

[10]:
```
%%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))

# Yash chopra nm0007181
query8 ="""
WITH common_query as (
SELECT Trim(actor) AS Actor, pactor.pid ap,
                director, pdir.pid dp
        FROM   m_cast mc
               INNER JOIN (SELECT m.mid
                           FROM   movie m) AS m
                       ON m.mid = Trim(mc.mid)
               INNER JOIN (SELECT md.pid,
                                  md.mid
                           FROM   m_director md) AS md
                       ON md.mid = Trim(mc.mid)
               INNER JOIN (SELECT p.pid,
                                  p.NAME AS actor
                           FROM   person p) AS pactor
                       ON pactor.pid = Trim(mc.pid)
               INNER JOIN (SELECT p.pid,
                                  p.NAME AS director
                           FROM   person p) AS pdir
                       ON pdir.pid = Trim(md.pid)
)
SELECT a.actor,
       a.count
FROM   (SELECT ap AS Actor,
               Count(dp)    AS COUNT
        FROM common_query
        WHERE  director LIKE '%Yash Chopra%'
        GROUP  BY ap
        ) a
       LEFT JOIN (SELECT actor,
                         Max(count) AS COUNT
                  FROM   (SELECT ap AS Actor, director ,
                                        Count(dp)    AS COUNT
                          FROM   common_query
                          WHERE  director NOT LIKE '%Yash Chopra%'
                          GROUP  BY actor,
```

```
                                    director)
                  GROUP  BY actor
                  ) b
        ON a.actor = b.actor

WHERE  a.count >= b.count
        OR b.actor IS NULL

ORDER  BY a.count DESC

"""
grader_8(query8)
```

```
       Actor  COUNT
0  nm0707271     11
1  nm0471443     10
2  nm0407002      9
3  nm0004434      7
4  nm0347901      5
5  nm0716851      5
6  nm0433945      4
7  nm0755087      4
8  nm0802183      4
9  nm0158332      3
(245, 2)
CPU times: user 670 ms, sys: 0 ns, total: 670 ms
Wall time: 668 ms
```

1.4.9 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.

[16]:
```
%%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 = """
with temp as (
    select
        distinct trim(m.mid) mid
    from movie m
    join m_cast mc on m.mid = mc.mid
```

```
    where lower(trim(pid)) = 'nm0451321'
),
-- all actors in the movies except shahrukh
temp2 as (
    select
        distinct trim(mc.pid) pid
    from temp t
    join m_cast mc on trim(t.mid) = trim(mc.mid)
    join person p on trim(p.pid)  = trim(mc.pid)
    where lower(trim(mc.pid)) != 'nm0451321'
),
temp3 as(
select
    distinct trim(mc.mid) mid
from m_cast mc
join temp2 as t on trim(mc.pid) = t.pid
where trim(mc.mid) not in ( select mid from temp )
)
-- selecting actors from that movies expect the ones
-- who worked with shahrukh
select
    distinct pid
from m_cast mc
join temp3 t on trim(mc.mid) = trim(t.mid)
where trim(mc.pid) not in ( select pid from temp2 )
"""
grader_9(query9)
```

```
         PID
0   nm2539953
1   nm0922035
2   nm0324658
3   nm0943079
4   nm0000218
5   nm0001394
6   nm0929654
7   nm3116102
8   nm3248891
9   nm2418809
(25698, 1)
CPU times: user 1min 17s, sys: 115 ms, total: 1min 17s
Wall time: 1min 17s
```