SQL Assignment

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

from IPython.display import display, HTML
```

In [2]:
```
# Note that this is not the same db we have used in course videos,
# https://drive.google.com/file/d/10-1-L1DdNxEK6O6nG2jS31MbrMh-OnXM
```

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

Overview of all tables

In [4]:
```
tables = pd.read_sql_query("SELECT NAME AS 'Table_Name' FROM sqlite
tables = tables["Table_Name"].values.tolist()
```

In [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

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(*)

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.

In [6]:
```
cursor = conn.cursor()
query1 = """ UPDATE Movie SET year = CAST(SUBSTR(TRIM(year),-4) AS
cursor.execute(query1)
```

Out[6]:
```
<sqlite3.Cursor at 0x7f584e63d810>
```

In [7]:
```
%%time
def grader_1(q1):
    q1_results  = pd.read_sql_query(q1,conn)
    print(q1_results.head(10))
    assert (q1_results.shape == (232,3))

query1 = """ select p.name, m.title, m.year from M_genre mg
             inner join Movie m on m.MID = mg.MID
             inner join Genre g on g.GID = mg.GID
             inner join M_Director md on m.mid = md.mid
             inner join Person p on p.pid = md.pid
             where (m.year % 4 == 0 or (m.year % 100 == 0 and m.yea
grader_1(query1)
```

```
                Name                                title  year
0       Milap Zaveri                            Mastizaade  2016
1       Danny Leiner  Harold & Kumar Go to White Castle  2004
2     Anurag Kashyap                 Gangs of Wasseypur  2012
3       Frank Coraci        Around the World in 80 Days  2004
4      Griffin Dunne             The Accidental Husband  2008
5        Anurag Basu                             Barfi!  2012
6    Gurinder Chadha                  Bride & Prejudice  2004
7         Mike Judge    Beavis and Butt-Head Do America  1996
8   Tarun Mansukhani                            Dostana  2008
9       Shakun Batra                       Kapoor & Sons  2016
CPU times: user 72.6 ms, sys: 2.91 ms, total: 75.5 ms
Wall time: 84.3 ms
```

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

In [8]:
```
cursor = conn.cursor()
query1 = """ UPDATE M_Cast SET pid = TRIM(pid) """
cursor.execute(query1)
```

Out[8]:
```
<sqlite3.Cursor at 0x7f584e3a1340>
```

In [9]:
```
%%time
def grader_2(q2):
    q2_results  = pd.read_sql_query(q2,conn)
    print(q2_results.head(10))
    print(q2_results.shape)
    assert (q2_results.shape == (17,1))


query2 = """ select p.name as 'Actor_Names' from M_Cast mc
             inner join Movie m on m.MID = mc.MID
             inner join Person p on p.pid = mc.pid
             where m.title like 'Anand' """
grader_2(query2)
# inner join Person p on p.pid = mc.pid
```

```
         Actor_Names
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
(17, 1)
CPU times: user 153 ms, sys: 7.05 ms, total: 160 ms
Wall time: 164 ms
```

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

In [10]:
```
%%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 In
)
) 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 In
)
) 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
```

```
(4942, 1)
(62570, 1)
True
CPU times: user 247 ms, sys: 7.9 ms, total: 254 ms
Wall time: 256 ms
```

In [11]:
```
%%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 = """ with
                first as
                (
                    select distinct trim(mc.PID) PID from Movie M j
                ),
                second AS
                (
                    select distinct trim(MC.PID) PID from Movie M j
                )
                select distinct trim(P.Name) Actor_Name FROM first
                join Person P on first.PID = P.PID  """

grader_3(query3)
```

```
         Actor_Name
0      Rishi Kapoor
1  Amitabh Bachchan
2            Asrani
3      Zohra Sehgal
4   Parikshat Sahni
5     Rakesh Sharma
6       Sanjay Dutt
7         Ric Young
8             Yusuf
9    Suhasini Mulay
CPU times: user 319 ms, sys: 11.2 ms, total: 330 ms
Wall time: 336 ms
```

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.

In [12]:
```
%%time

def grader_4a(query_4a):
    query_4a = pd.read_sql_query(query_4a,conn)
    print(query_4a)
    return (query_4a.shape == (1462,2))

query_4a =""" select p.Name, count(1) from M_Director md join Perso
print(grader_4a(query_4a))

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

```
                    Name  count(1)
0            David Dhawan        39
1            Mahesh Bhatt        35
2            Priyadarshan        30
3         Ram Gopal Varma        30
4            Vikram Bhatt        29
...                   ...       ...
1457         Deb Medhekar         1
1458      Rahi Anil Barve         1
1459  Sudip Bandyopadhyay         1
1460   Manu Prakash Singh         1
1461         Vinod Tiwari         1

[1462 rows x 2 columns]
True
CPU times: user 65.1 ms, sys: 1.03 ms, total: 66.1 ms
Wall time: 66.3 ms
```

In [13]:
```
%%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, count(1) from M_Director md join Person
grader_4(query4)
```

```
                    Name  count(1)
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 61.7 ms, sys: 997 µs, total: 62.7 ms
Wall time: 62.9 ms
```

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

In [14]:
```
cursor = conn.cursor()
query1 = """ UPDATE M_Cast SET pid = TRIM(pid) """
cursor.execute(query1)
```

Out[14]:
```
<sqlite3.Cursor at 0x7f584e63dce0>
```

In [15]:
```
%%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)
    return (query_5aa.shape == (8846,3))

query_5aa = """ select distinct mc.mid, p.gender, count(2) from M_C
                inner join Person p on trim(mc.pid) = p.pid group b
print(grader_5aa(query_5aa))

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

query_5ab = """ select distinct mc.mid, p.gender, count(2) from M_C
                inner join Person p on trim(mc.pid) = p.pid where p
                 """
print(grader_5ab(query_5ab))

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

```
            MID  Gender  count(2)
0     tt0021594    None         1
1     tt0021594  Female         3
2     tt0021594    Male         5
3     tt0026274    None         2
4     tt0026274  Female        11
...         ...     ...       ...
8841  tt8932884  Female         1
8842  tt8932884    Male         2
8843  tt9007142    None         1
8844  tt9007142  Female         4
8845  tt9007142    Male         5

[8846 rows x 3 columns]
True
         MID Gender  count(2)
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 364 ms, sys: 2.95 ms, total: 367 ms
Wall time: 369 ms
```

In [16]:

```
%%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 m.year, count(*) as Female_Cast_Only_Movies fr
              (select distinct mid from M_Cast where mid not in
                (select distinct mc.mid from M_Cast mc
                inner join Person p on trim(mc.pid) = p.pid where p
                on m.mid = t.mid group by 1
                """
grader_5a(query5a)
```

```
   year  Female_Cast_Only_Movies
0  1939                        1
1  1999                        1
2  2000                        1
3  2018                        1
CPU times: user 198 ms, sys: 1.87 ms, total: 200 ms
Wall time: 201 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.

In [17]:
```
%%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(t1.count as float)/CAST(count(m.m
              (select m.year, count(*) as count from Movie m join
                (select distinct mid from M_Cast where mid not in
                  (select distinct mc.mid from M_Cast mc
                  inner join Person p on trim(mc.pid) = p.pid where
                  on m.mid = t.mid group by 1) as t1 on m.year = t1

query5a = """ select m.year """
grader_5b(query5b)
```

```
   year  Percentage_Female_Only_Movie  Total_Movies
0  1939                      0.500000             2
1  1999                      0.015152            66
2  2000                      0.015625            64
3  2018                      0.009615           104
CPU times: user 182 ms, sys: 2 ms, total: 184 ms
Wall time: 185 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.

In [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 = """ select title, count from ( select distinct mc.mid, m.t
            from M_Cast mc join Movie m on m.mid = mc.mid group by
grader_6(query6)
```

```
                         title  count
0                Ocean's Eight    238
1                     Apaharan    233
2                         Gold    215
3              My Name Is Khan    213
4  Captain America: Civil War    191
5                     Geostorm    170
6                      Striker    165
7                         2012    154
8                       Pixels    144
9       Yamla Pagla Deewana 2    140
CPU times: user 112 ms, sys: 5.91 ms, total: 118 ms
Wall time: 119 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

In [19]:
```
%%time
def grader_7a(q7a):
    q7a_results  = pd.read_sql_query(q7a,conn)
    print(q7a_results.head(10))
    print(q7a_results.shape)
    assert (q7a_results.shape == (78, 2))
#*** Write a query that computes number of movies in each year ***
query7a = """ select distinct m.year Movie_Year, count(*) as Total_
grader_7a(query7a)

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

```
  Movie_Year  Total_Movies
0       2018           104
1       2017           126
2       2016           129
3       2015           119
4       2014           126
5       2013           136
6       2012           110
7       2011           116
8       2010           125
9       2009           109
(78, 2)
CPU times: user 11.2 ms, sys: 0 ns, total: 11.2 ms
Wall time: 11.3 ms
```

In [20]:
```
%%time
def grader_7b(q7b):
    q7b_results  = pd.read_sql_query(q7b,conn)
    print(q7b_results.head(10))
    print(q7b_results.shape)
    assert (q7b_results.shape == (713, 4))
# ***
#     Write a query that will do joining of the above table(7a) wit
#     such that you will join with only rows if the second tables y
# ***
query7b = """ with data as (select distinct m.year, count(*) as cou

              select a.year as Movie_Year, a.count as Movie_Count,
              select a.year as Movie_Year, a.count as Movie_Count,
          """
grader_7b(query7b)
# if you see the below results the first movie year is less than 2n
# 2nd movie year is less or equal to the first movie year+9

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

```
  Movie_Year  Movie_Count Movie_Year  Movie_Count
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
(713, 4)
CPU times: user 18.1 ms, sys: 0 ns, total: 18.1 ms
Wall time: 18.3 ms
```

In [21]:
```
%%time
def grader_7(q7):
    q7_results  = pd.read_sql_query(q7,conn)
    print(q7_results.head(15))
    print(q7_results.shape)
    assert (q7_results.shape == (1, 2))

#*** Write a query that will return the decade that has maximum num
query7 = """
              with data as (select distinct m.year, count(*) as cou

              list as
              (select a.year as year_st, a.count as count1, b.year
              select a.year as year_st, a.count as count1, b.year a

              select '2008-2017' year_start, max(count) 'Decade_Mov
         """
grader_7(query7)
# if you check the output we are printinng all the year in that dec
```

```
  year_start  Decade_Movie_Count
0  2008-2017                1203
(1, 2)
CPU times: user 14.5 ms, sys: 0 ns, total: 14.5 ms
Wall time: 14.6 ms
```

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

In [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 = """ select mc.pid as actor, md.pid as director, count(*)
              join M_Director md on mc.mid = md.mid group by 1,2 or
grader_8a(query8a)

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

```
       actor   director  movies
0  nm0456094  nm0223522      23
1  nm0007106  nm0223522      20
2  nm0434318  nm0223522      20
3  nm0318622  nm0080315      19
4  nm0332871  nm0223522      17
5  nm0712546  nm0698184      16
6  nm2147526  nm0698184      15
7  nm0442479  nm0223522      14
8  nm0451272  nm0080315      14
9  nm0451600  nm0223522      14
CPU times: user 275 ms, sys: 12 ms, total: 287 ms
Wall time: 289 ms
```

In [23]:
```
cursor = conn.cursor()
query1 = """ UPDATE M_Cast SET pid = trim(pid) """
cursor.execute(query1)
```

Out[23]:
```
<sqlite3.Cursor at 0x7f584e63ddc0>
```

In [27]:
```
%%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 = """ WITH
        yashchopra AS(
              SELECT (mc.PID) AS Actor_Id, (md.PID) AS Director_Id,
              GROUP BY mc.PID,md.PID),
        yashchopra_movie AS (
              SELECT Count(md.mid) AS Count FROM M_Cast AS mc JOIN
            GROUP BY mc.PID, md.PID ORDER BY Count(*) DESC LIMIT 1)

        SELECT p.name, Movie_count FROM yashchopra JOIN Person AS p
        AND yashchopra.Director_Id IN (SELECT PID FROM Person WHERE


grader_8(query8)
```

```
                Name  Movie_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 3.35 s, sys: 180 ms, total: 3.53 s
Wall time: 3.55 s
```

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.

In [25]:
```
%%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 = """ with
              s0 as (
                select trim(pid) as pid from Person where trim(Name
              ),
              s1_m as (
                select distinct mc.mid mid, s0.pid pid from M_Cast
              ),
              s1_a as (
                select distinct trim(mc.pid) pid from M_Cast mc, s1
                except select trim(pid) as pid from Person where tr
              ),
              s2_m as (
                select distinct trim(mc.mid) mid, s1_a.pid pid from
              ),
              s2_a as (
                select pid from (select distinct trim(mc.pid) pid f
                except select trim(pid) as pid from Person where tr
              )

              select pid S1_PID from s1_a
                  """

grader_9a(query9a)
# using the above query, you can write the answer to the given ques

# selecting actors who acted with srk (S1)
# selecting all movies where S1 actors acted, this forms S2 movies
# selecting all actors who acted in S2 movies, this gives us S2 act
# removing S1 actors from the combined list of S1 & S2 actors, so t
```

```
      S1_PID
0  nm0000818
1  nm0000821
2  nm0001934
3  nm0002043
4  nm0004109
5  nm0004334
6  nm0004335
7  nm0004363
8  nm0004418
9  nm0004429
(2382, 1)
CPU times: user 55.1 ms, sys: 3.95 ms, total: 59.1 ms
Wall time: 59.3 ms
```

In [26]:
```
%%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
              s0 as (
                select trim(pid) as pid from Person where trim(Name
              ),
              s1_m as (
                select distinct mc.mid mid, s0.pid pid from M_Cast
              ),
              s1_a as (
                select distinct trim(mc.pid) pid from M_Cast mc, s1
                except select trim(pid) as pid from Person where tr
              ),
              s2_m as (
                select distinct trim(mc.mid) mid, s1_a.pid pid from
              ),
              s2_a as (
                select pid from (select distinct trim(mc.pid) pid f
                except select trim(pid) as pid from Person where tr
              )

              select p.name from s2_a join Person p on s2_a.pid = p
                 """
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)
CPU times: user 857 ms, sys: 10.9 ms, total: 868 ms
Wall time: 873 ms
```

In [26]:
```
```