SQL Assignment

In [27]:

```
import pandas as pd
import sqlite3

from IPython.display import display, HTML
```

In [ ]:

```
# Note that this is not the same db we have used in course videos, please download from thi
# https://drive.google.com/file/d/1O-1-L1DdNxEK6O6nG2jS31MbrMh-OnXM/view?usp=sharing
```

In [28]:

```
conn = sqlite3.connect("Db-IMDB-Assignment.db")
```

Overview of all tables

In [29]:

```
tables = pd.read_sql_query("SELECT NAME AS 'Table_Name' FROM sqlite_master WHERE type='tabl
tables = tables["Table_Name"].values.tolist()
```

In [30]:

```
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 [10]:

```
%%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 m.title, p.name, m.year
             FROM Movie m JOIN
             M_Director d
             ON m.MID = d.MID JOIN
             Person p
             ON d.PID = P.PID JOIN
             M_genre mg
             ON m.MID = mg.MID JOIN
             Genre g
             ON g.GID = mg.GID
             WHERE g.name LIKE '%Comedy%' AND (CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER)%4 =
             ( CAST(SUBSTR(TRIM(m.year),-4) AS INTEGER)%100 <>0 OR CAST(SUBSTR(TRIM(m.year)

"""
grader_1(query1)
```

```
                                title              Name  year
0                          Mastizaade      Milap Zaveri  2016
1  Harold & Kumar Go to White Castle      Danny Leiner  2004
2                    Gangs of Wasseypur    Anurag Kashyap  2012
3             Around the World in 80 Days      Frank Coraci  2004
4                 The Accidental Husband     Griffin Dunne  2008
5                                Barfi!       Anurag Basu  2012
6                      Bride & Prejudice   Gurinder Chadha  2004
7       Beavis and Butt-Head Do America        Mike Judge  1996
8                               Dostana  Tarun Mansukhani  2008
9                          Kapoor & Sons      Shakun Batra  2016
Wall time: 350 ms
```

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

In [11]:

```
%%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 Name FROM Person WHERE TRIM(PID) IN ( SELECT TRIM(PID) FROM M_Cast WHER
            (SELECT TRIM(MID) FROM Movie WHERE title = 'Anand')) """
grader_2(query2)
```

```
                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
Wall time: 318 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 [12]:

```
%%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 distinct(p.Name)
from Person p
join
(
    select trim(mc.PID) PD 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
join
(
    select trim(mc.PID) PD 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
```

```
(1937, 1)
(62570, 1)
False
Wall time: 576 ms
```

In [31]:

```
%%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 FROM  Person
WHERE PID IN (
Select distinct p.PID
from Person p
join
(
    select trim(mc.PID) PD 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

WHERE p.PID IN(

Select distinct p.PID from Person p
join
(
    select trim(mc.PID) PD 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
))
"""
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
Wall time: 568 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 [14]:

```
%%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, COUNT(MID)  FROM M_Director GROUP BY TRIM(PID)  """
print(grader_4a(query_4a))

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

```
         PID  COUNT(MID)
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: 95.3 ms
```

In [15]:

```
%%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(md.MID)  FROM Person p JOIN
             M_Director md
             ON  md.PID = p.PID
             group BY TRIM(md.PID) HAVING COUNT(md.MID)>= 10 ORDER BY COUNT(md.MID) DESC ""

grader_4(query4)
```

```
                   Name  COUNT(md.MID)
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
Wall time: 56 ms
```

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

In [16]:

```
%%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 TRIM(mc.MID), p.Gender, count(*) from M_Cast mc
                JOIN Person p
                ON p.PID = TRIM(mc.PID)
                GROUP BY mc.MID,p.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 TRIM(mc.MID), p.Gender, count(*) from M_Cast mc
                JOIN Person p
                ON p.PID = TRIM(mc.PID)
                GROUP BY mc.MID,p.Gender
                Having  p.Gender = 'Male' """

print(grader_5ab(query_5ab))


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

```
  TRIM(mc.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
  TRIM(mc.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: 747 ms
```

In [17]:

```
%%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
                CAST(SUBSTR(M.year,-4) AS UNSIGNED) year,
                COUNT(DISTINCT TRIM(MID) ) FEMALE_MOVIES
                FROM
                Movie M
                WHERE
                TRIM(MID) NOT IN (SELECT TRIM(mc.MID) from M_Cast mc
                JOIN Person p
                ON p.PID = TRIM(mc.PID)
                WHERE
                TRIM(P.Gender) IN ('Male','None'))

                GROUP BY
                CAST(SUBSTR(M.year,-4) AS UNSIGNED)
                ORDER BY
                year
                 """
grader_5a(query5a)
```

```
   year  FEMALE_MOVIES
0  1939              1
1  1999              1
2  2000              1
3  2018              1
Wall time: 488 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 [18]:
```

```
%%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 movie.year, count(movie.mid) as movie_per_year,cast(r1.female_cast as r
            inner join
            (
            SELECT Movie.year as Year, COUNT(Movie.mid) AS female_cast
            FROM Movie
            WHERE Movie.MID NOT IN (
            SELECT Movie.MID from Movie
            Inner Join M_cast
            on TRIM(M_cast.MID) = Movie.MID
            Inner Join Person
            on TRIM(M_cast.PID) = Person.PID
            WHERE Person.Gender!='Female'
            GROUP BY Movie.MID
              )
            GROUP BY Movie.year
            Order By Movie.year asc
            ) r1
            on r1.year = movie.year
            GROUP BY movie.year
            ORDER BY movie.year"""

grader_5b(query5b)
```

```
     year  movie_per_year  percentage
0    1939               2    0.500000
1    1999              66    0.015152
2    2000              64    0.015625
3  I 2018              10    0.100000
Wall time: 400 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 [19]:

```
%%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, count(MC.PID) Count FROM movie M
            JOIN M_Cast MC
            ON MC.MID = M.MID
            group By MC.MID  ORDER BY  count(MC.PID) DESC"""
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
Wall time: 376 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 [20]:

```
%%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 Integer) year, count(title) FROM MOVIE GROUP B
grader_7a(query7a)

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

```
   year  count(title)
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: 16 ms
```

In [21]:

```
%%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 CAST(SUBSTR(m1.year,-4) AS Integer) movie_year, count(m1.title) Total_m
        FROM Movie m1
        JOIN Movie m2
        ON   CAST(SUBSTR(m2.year,-4) AS Integer) <= CAST(SUBSTR(m1.year,-4) AS Integer) + 9
        GROUP BY CAST(SUBSTR(m1.year,-4) AS Integer)"""
grader_7b(query7b)
# if you see the below results the first movie year is less than 2nd movie year and
# 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 question
```

```
   movie_year  Total_movies  movie_year  total_movies
0        1931             6        1939             6
1        1936            24        1939            24
2        1939            30        1939            30
3        1941            20        1939            20
4        1943            32        1939            32
5        1946           110        1939           110
6        1947           122        1939           122
7        1948           222        1939           222
8        1949           249        1939           249
9        1950           178        1939           178

---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<timed exec> in <module>()

<timed exec> in grader_7b(q7b)

AssertionError: 
```

In [22]:

```
%%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 r1.year as decade,
             count(*) as total_movies
             from (select distinct year from Movie) r1 join
             Movie m
             on m.year >= r1.year and m.year < r1.year + 10
             group by r1.year
             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 pr
```

```
  decade  total_movies
0   2008          1126
Wall time: 120 ms
```

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

In [23]:

```
%%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 md.PID director, mc.PID actor, count(*) FROM M_Director md
             JOIN M_Cast mc
             ON mc.MID = md. MID
             group By mc.PID, md.PID"""
grader_8a(query8a)

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

```
    director      actor  count(*)
0  nm0496746  nm0000002         1
1  nm0000180  nm0000027         1
2  nm0896533  nm0000039         1
3  nm0896533  nm0000042         1
4  nm0004292  nm0000047         1
5  nm0485943  nm0000073         1
6  nm0000229  nm0000076         1
7  nm0178997  nm0000092         1
8  nm0000269  nm0000093         1
9  nm0113819  nm0000096         1
Wall time: 800 ms
```

In [24]:

```
%%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 actorName, yash_chopra_movies from
(SELECT * FROM
(SELECT Person.Name actorName,M_Cast.PID actor,M_Director.PID director, COUNT(*) yash_chopr
Movie m
INNER JOIN M_Director ON M_Director.MID=m.MID
INNER JOIN M_Cast ON m.MID=TRIM(M_Cast.MID)
INNER JOIN Person ON TRIM(M_Cast.PID)= Person.PID
GROUP BY M_Cast.PID,M_Director.PID
HAVING director =
(
SELECT PID FROM PERSON p WHERE TRIM(Name) like '%Yash Chopra%'
))
yash LEFT JOIN
(
SELECT actor, MAX(movie_count)max_movie_count FROM
(
SELECT M_Cast.PID actor,M_Director.PID director, COUNT(*) movie_count FROM Movie m1
INNER JOIN M_Director ON M_Director.MID = m1.MID
INNER JOIN M_Cast ON m1.MID=TRIM(M_Cast.MID)
GROUP BY M_Cast.PID,M_Director.PID
)
GROUP BY actor
)all_actor
ON yash.actor= all_actor.actor where yash_chopra_movies>=max_movie_count
)"""

grader_8(query8)
```

```
          actorName  yash_chopra_movies
0     Shashi Kapoor                   7
1       Yash Chopra                   2
2    Akhtar-Ul-Iman                   1
3         Murad Ali                   1
4      Badri Prasad                   1
5        Saira Banu                   1
6        Raj Bharti                   1
7     Ashwini Bhave                   1
8   Andrew Bicknell                   1
9    Paul Blackwell                   1
(245, 2)
Wall time: 1.19 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 = """SELECT DISTINCT TRIM(mc.PID) FROM M_Cast mc WHERE TRIM(mc.PID) != 'nm0451321'
            (SELECT TRIM(mc.MID) FROM M_Cast mc  WHERE TRIM(mc.PID) IN
            (SELECT PID FROM Person WHERE Name like '%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 actor
```

```
  TRIM(mc.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: 97.4 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 = """ SELECT Name FROM Person WHERE  PID IN
            (SELECT DISTINCT TRIM(mc1.PID)  FROM M_Cast mc1 WHERE TRIM(mc1.MID) IN
            (SELECT DISTINCT TRIM(mc1.MID) FROM M_Cast mc1 WHERE TRIM(mc1.PID) IN
            (SELECT DISTINCT TRIM(mc.PID) FROM M_Cast mc WHERE  TRIM(mc.MID) IN
            (SELECT DISTINCT TRIM(mc.MID) FROM M_Cast mc  WHERE TRIM(mc.PID) IN
            (SELECT PID FROM Person WHERE Name like '%Shah Rukh Khan%')))) AND TRIM(mc1.PID
            SELECT DISTINCT TRIM(mc.PID) FROM M_Cast mc WHERE  TRIM(mc.MID) IN
            (SELECT TRIM(mc.MID) FROM M_Cast mc  WHERE TRIM(mc.PID) IN
            (SELECT PID FROM Person WHERE Name like '%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: 576 ms
```