Importing required packages

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

Setting up connection to database

In [2]:
```
conn = sqlite3.connect('Db-IMDB.db')
```

Test Query

In [3]:
```
pd.read_sql_query('SELECT * FROM Movie LIMIT 10', conn)
```

Out[3]:

	index	MID	title	year	rating	num_votes
0	0	tt2388771	Mowgli	2018	6.6	21967
1	1	tt5164214	Ocean's Eight	2018	6.2	110861
2	2	tt1365519	Tomb Raider	2018	6.4	142585
3	3	tt0848228	The Avengers	2012	8.1	1137529
4	4	tt8239946	Tumbbad	2018	8.5	7483
5	5	tt7027278	Kedarnath	2018	5.5	1970
6	6	tt3498820	Captain America: Civil War	2016	7.8	536641
7	7	tt8108198	Andhadhun	2018	9.0	18160
8	8	tt3741834	Lion	2016	8.1	170216
9	9	tt6747420	Rajma Chawal	2018	5.7	681

Question 1:

After Checking some queries I got to see some years having roman numbers in front of them

In [4]:
```
query = """
SELECT DISTINCT year
FROM Movie
WHERE CAST(year AS INT)=0
LIMIT 10
"""
pd.read_sql_query(query, conn)
```

Out[4]:

	year
0	I 2009
1	I 2018
2	XVII 2016
3	I 2017
4	II 2018
5	I 2002
6	III 2016
7	I 2015
8	I 2016
9	I 2006

To Correct them I took last chars and turned them to integer

In [5]:
```
# Took code from src: https://stackoverflow.com/questions/12504985/how-to-take-last-four-characters-from-a-varch
query = """
SELECT DISTINCT year, CAST(SUBSTR(year, LENGTH(year)-3, 4) AS INT) real_year
FROM Movie
WHERE CAST(year AS INT)=0
LIMIT 10
"""
pd.read_sql_query(query, conn)
```

Out[5]:

	year	real_year
0	I 2009	2009
1	I 2018	2018
2	XVII 2016	2016
3	I 2017	2017
4	II 2018	2018
5	I 2002	2002
6	III 2016	2016
7	I 2015	2015
8	I 2016	2016
9	I 2006	2006

In [6]:
```
year_formula = 'CAST(SUBSTR(year, LENGTH(year)-3, 4) AS INT)'
```

From now I use above formula for year when its needed in this notebook

In [7]:
```
query = f"""
SELECT DISTINCT {year_formula} real_year
FROM Movie
WHERE CAST(real_year AS INT)=0
"""
pd.read_sql(query, conn)
```

Out[7]:

	real_year

In above statement, I used string formatting to insert year formula and aliased it as real_year. As you can see above there are no years which give wrong result now.

Now checking movie names and years for 'Comedy' genre and having a leap year

In [8]:
```
# Checking Leap year code is taken from StackOverFlow
# src: https://stackoverflow.com/questions/6534788/check-for-leap-year
query = f"""
SELECT m.title, {year_formula} real_year
FROM Movie m JOIN M_Genre mg ON m.MID = mg.MID
WHERE ((real_year%4 = 0 AND real_year%100 <> 0) OR real_year%400 = 0)
AND mg.GID IN (
SELECT GID FROM Genre
WHERE Name LIKE '%Comedy%'
)
"""
pd.read_sql_query(query, conn)
```

Out[8]:

	title	real_year
0	Mastizaade	2016
1	Harold & Kumar Go to White Castle	2004
2	Gangs of Wasseypur	2012
3	Around the World in 80 Days	2004
4	The Accidental Husband	2008
5	Barfi!	2012
6	Bride & Prejudice	2004
7	Beavis and Butt-Head Do America	1996
8	Dostana	2008
9	Kapoor & Sons	2016
10	Rab Ne Bana Di Jodi	2008
11	Dishoom	2016
12	Baar Baar Dekho	2016
13	Oye Lucky! Lucky Oye!	2008
14	OMG: Oh My God!	2012
15	Befikre	2016
16	Student of the Year	2012
17	Main Hoon Na	2004

	title	real_year
18	Vicky Donor	2012
19	Jaane Tu... Ya Jaane Na	2008
20	Hera Pheri	2000
21	Hotel Salvation	2016
22	The Other End of the Line	2008
23	Cocktail	2012
24	Tere Mere Sapne	1996
25	Happy Bhag Jayegi	2016
26	English Vinglish	2012
27	Eega	2012
28	A Flying Jatt	2016
29	Freaky Ali	2016
...	...	...
202	Luv U Soniyo	2012
203	Daal Mein Kuch Kaala Hai	2012
204	Kis Kis Ki Kismat	2004
205	New Delhi	1956
206	Ab Ayega Mazaa	1984
207	Dadar Kirti	1980
208	Bach ke Zara	2008
209	Ek Hota Vidushak	1992
210	Maan Gaye Mughall-E-Azam	2008
211	Ugly Aur Pagli	2008
212	Bol Radha Bol	1992
213	Ghar Ghar Ki Kahani	1988
214	Suno Sasurjee	2004
215	Paisa Vasool	2004

	title	real_year
216	Hari Om	2004
217	Do Dooni Char	1968
218	Shaadi Ka Laddoo	2004
219	Popcorn Khao! Mast Ho Jao	2004
220	Ginny Aur Johny	1976
221	Lakhon Ki Baat	1984
222	Khokababu	2012
223	Meerabai Not Out	2008
224	Ranga S.S.L.C	2004
225	Yaad Rakhegi Duniya	1992
226	Pestonjee	1988
227	Let's Enjoy	2004
228	Sathyam	2008
229	Tandoori Love	2008
230	Le Halua Le	2012
231	Raja Aur Rangeeli	1996

232 rows × 2 columns

We got 232 rows of movies which released in leap year and are comedy. Now building query for our **FINAL ANSWER**.

In [9]:
```
query = f"""
SELECT DISTINCT TRIM(p.Name) director_name, req_movies.title movie_title, req_movies.real_year year
FROM Person p JOIN M_Director md ON p.PID = md.PID
  JOIN
  (
    SELECT m.MID mid, m.title title, {year_formula} real_year FROM Movie m
    JOIN M_Genre mg ON m.MID = mg.MID
    WHERE ((real_year%4 = 0 AND real_year%100 <> 0) OR real_year%400 = 0)
    AND mg.GID IN (
      SELECT GID FROM Genre
      WHERE Name LIKE '%Comedy%'
    )
  ) req_movies ON md.MID = req_movies.mid
"""
pd.read_sql_query(query, conn)
```

Out[9]:

	director_name	movie_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
10	Aditya Chopra	Rab Ne Bana Di Jodi	2008
11	Rohit Dhawan	Dishoom	2016
12	Nitya Mehra	Baar Baar Dekho	2016
13	Dibakar Banerjee	Oye Lucky! Lucky Oye!	2008
14	Umesh Shukla	OMG: Oh My God!	2012
15	Aditya Chopra	Befikre	2016

	director_name	movie_title	year
16	Karan Johar	Student of the Year	2012
17	Farah Khan	Main Hoon Na	2004
18	Shoojit Sircar	Vicky Donor	2012
19	Abbas Tyrewala	Jaane Tu... Ya Jaane Na	2008
20	Priyadarshan	Hera Pheri	2000
21	Shubhashish Bhutiani	Hotel Salvation	2016
22	James Dodson	The Other End of the Line	2008
23	Homi Adajania	Cocktail	2012
24	Joy Augustine	Tere Mere Sapne	1996
25	Mudassar Aziz	Happy Bhag Jayegi	2016
26	Gauri Shinde	English Vinglish	2012
27	S.S. Rajamouli	Eega	2012
28	Remo D'Souza	A Flying Jatt	2016
29	Sohail Khan	Freaky Ali	2016
...	...	...	...
202	Joe Rajan	Luv U Soniyo	2012
203	Anand Balraj	Daal Mein Kuch Kaala Hai	2012
204	Govind Menon	Kis Kis Ki Kismat	2004
205	Mohan Segal	New Delhi	1956
206	Pankaj Parashar	Ab Ayega Mazaa	1984
207	Tarun Majumdar	Dadar Kirti	1980
208	Salim Raza	Bach ke Zara	2008
209	Jabbar Patel	Ek Hota Vidushak	1992
210	Sanjay Chhel	Maan Gaye Mughall-E-Azam	2008
211	Sachin Kamlakar Khot	Ugly Aur Pagli	2008
212	David Dhawan	Bol Radha Bol	1992
213	Kalpataru	Ghar Ghar Ki Kahani	1988

	director_name	movie_title	year
214	Vimal Kumar	Suno Sasurjee	2004
215	Srinivas Bhashyam	Paisa Vasool	2004
216	Ganapathy Bharat	Hari Om	2004
217	Debu Sen	Do Dooni Char	1968
218	Raj Kaushal	Shaadi Ka Laddoo	2004
219	Kabir Sadanand	Popcorn Khao! Mast Ho Jao	2004
220	Mehmood	Ginny Aur Johny	1976
221	Basu Chatterjee	Lakhon Ki Baat	1984
222	Shankaraiya	Khokababu	2012
223	Chandrakant Kulkarni	Meerabai Not Out	2008
224	Yograj Bhat	Ranga S.S.L.C	2004
225	Deepak Anand	Yaad Rakhegi Duniya	1992
226	Vijaya Mehta	Pestonjee	1988
227	Siddharth Anand Kumar	Let's Enjoy	2004
228	Amma Rajasekhar	Sathyam	2008
229	Oliver Paulus	Tandoori Love	2008
230	Raja Chanda	Le Halua Le	2012
231	K.S. Prakash Rao	Raja Aur Rangeeli	1996

232 rows × 3 columns

Got 232 Rows. Used Trim in above query as they are duplicate rows in Person table.

Question 2:

Checking if there are several movies named 'Anand'

In [10]:
```
query = """
SELECT * FROM Movie WHERE title='Anand'
"""
pd.read_sql_query(query, conn)
```

Out[10]:

	index	MID	title	year	rating	num_votes
0	66	tt0066763	Anand	1971	8.8	21616

As there is only one movie we can build the **FINAL QUERY**

In [11]:
```
query = """
SELECT Name FROM Person
WHERE PID IN (
SELECT TRIM(mc.PID)
FROM M_Cast mc JOIN Movie m ON m.MID = mc.MID
WHERE m.title='Anand'
)
"""
pd.read_sql_query(query, conn)
```

Out[11]:

	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
10	Brahm Bhardwaj
11	Gurnam Singh
12	Lalita Pawar
13	Durga Khote
14	Dara Singh
15	Johnny Walker
16	Moolchand

Got 17 Actors for movie 'Anand'

Question 3:

FINAL QUERY

In [12]:
```
query = f"""
SELECT DISTINCT TRIM(Name) Actor FROM Person
WHERE PID IN (
  SELECT TRIM(PID) FROM M_Cast
  WHERE MID IN (
    SELECT TRIM(MID) FROM Movie
    WHERE {year_formula} < 1970
  )
)

INTERSECT

SELECT DISTINCT TRIM(Name) Actor FROM Person
WHERE PID IN (
  SELECT TRIM(PID) FROM M_Cast
  WHERE MID IN (
    SELECT TRIM(MID) FROM Movie
    WHERE {year_formula} > 1990
  )
)
"""
pd.read_sql_query(query, conn)
```

Out[12]:

	Actor
0	A.K. Hangal
1	Aachi Manorama
2	Abbas
3	Abdul
4	Abhi Bhattacharya
5	Achala Sachdev
6	Adil
7	Ajay
8	Ajit
9	Akashdeep
10	Akbar Bakshi

	Actor
11	Alka
12	Allu Ramalingaiah
13	Altaf
14	Amar
15	Amarnath
16	Ameer
17	Amitabh Bachchan
18	Amjad Khan
19	Amol Sen
20	Amrit
21	Anand
22	Anand Kumar
23	Anand Tiwari
24	Anil
25	Anil Kumar
26	Anil Nagrath
27	Anjali Kadam
28	Anju Mahendru
29	Anoop Kumar
...	...
394	Tej Sapru
395	Thapa
396	Tulsi
397	Uma
398	Umesh Sharma
399	Unni Mary
400	Urmila Bhatt

	Actor
401	Usha Kiran
402	Utpal Dutt
403	Veena
404	Veera
405	Vijay
406	Vijayalalitha
407	Vijayalaxmi
408	Viju Khote
409	Vikram Makandar
410	Vineet Kumar
411	Vinod Mehra
412	Vinod Sharma
413	Vishnu
414	Vishwa Mehra
415	Waheeda Rehman
416	Wasi Khan
417	Yash Kumar
418	Yasmin
419	Yunus Parvez
420	Yusuf
421	Zia
422	Zohra Sehgal
423	Zul Vellani

424 rows × 1 columns

Got 424 Actors who acted before 1970 and after 1990.

Question 4:

FINAL QUERY

In [13]:
```
query = """
SELECT DISTINCT TRIM(p.Name) director, temp.movies no_of_movies
FROM Person p JOIN (
  SELECT PID, COUNT(MID) movies FROM M_Director
  GROUP BY PID
  HAVING movies>=10
) temp ON p.PID = temp.PID
ORDER BY no_of_movies DESC
"""
pd.read_sql_query(query, conn)
```

Out[13]:

	director	no_of_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
10	Abbas Alibhai Burmawalla	17
11	Shyam Benegal	17
12	Rama Rao Tatineni	17
13	Manmohan Desai	16
14	Gulzar	16
15	Raj N. Sippy	16
16	Mahesh Manjrekar	15
17	Raj Kanwar	15
18	Rajkumar Santoshi	14

	director	no_of_movies
19	Rahul Rawail	14
20	Raj Khosla	14
21	Indra Kumar	14
22	Anurag Kashyap	13
23	Rakesh Roshan	13
24	Ananth Narayan Mahadevan	13
25	Dev Anand	13
26	Vijay Anand	13
27	K. Raghavendra Rao	13
28	Harry Baweja	13
29	Satish Kaushik	12
30	Madhur Bhandarkar	12
31	Prakash Jha	12
32	Rohit Shetty	12
33	Anees Bazmee	12
34	Anil Sharma	12
35	Nagesh Kukunoor	12
36	Prakash Mehra	12
37	Guddu Dhanoa	12
38	Umesh Mehra	12
39	Ketan Mehta	11
40	Mohit Suri	11
41	Sanjay Gupta	11
42	Nasir Hussain	11
43	Pramod Chakravorty	11
44	Govind Nihalani	11
45	Tigmanshu Dhulia	10

	director	no_of_movies
46	Raj Kapoor	10
47	Sudhir Mishra	10
48	N. Chandra	10
49	Vishal Bhardwaj	10
50	Hansal Mehta	10
51	J.P. Dutta	10
52	J. Om Prakash	10
53	Bimal Roy	10
54	Mehul Kumar	10
55	K. Muralimohana Rao	10
56	Pankaj Parashar	10
57	K. Bapaiah	10

Got 58 directors who directed 10 movies or more.

Question 5a:

I Find that there are only 42 movies in 1990 where in question it is shown to be 13522. I Assume that the data is changed (reduced) because total number of movie Id's in our database is only 3475.

In [14]:
```
query = f"""
SELECT {year_formula} real_year, COUNT(MID) FROM Movie
WHERE real_year=1990
GROUP BY real_year
"""
pd.read_sql_query(query, conn)
```

Out[14]:

	real_year	COUNT(MID)
0	1990	42

In [15]:
```
query = """
SELECT COUNT(MID) FROM Movie
"""
pd.read_sql_query(query, conn)
```

Out[15]:

	COUNT(MID)
0	3475

Below query gets count of female cast for each movie

In [16]:
```
query = """
SELECT MID, COUNT(TRIM(PID)) female_cast FROM M_Cast
WHERE TRIM(PID) IN (
SELECT TRIM(PID) FROM Person
WHERE TRIM(Gender)='Female'
)
GROUP BY MID
"""
pd.read_sql_query(query, conn)
```

Out[16]:

	MID	female_cast
0	tt0021594	3
1	tt0026274	11
2	tt0027256	5
3	tt0028217	3
4	tt0031580	17
5	tt0033616	7
6	tt0036077	3
7	tt0038491	3
8	tt0039654	6
9	tt0040067	5
10	tt0041123	4
11	tt0041161	5
12	tt0041619	6
13	tt0043078	7
14	tt0043306	8
15	tt0043307	3
16	tt0043456	12
17	tt0044318	6
18	tt0044392	5
19	tt0044527	4

	MID	female_cast
20	tt0044761	3
21	tt0044769	8
22	tt0045467	4
23	tt0045506	4
24	tt0045529	5
25	tt0045693	16
26	tt0046164	12
27	tt0046427	15
28	tt0046673	6
29	tt0046703	3
...	...	...
3420	tt7853242	11
3421	tt7856176	2
3422	tt7881542	3
3423	tt7919680	9
3424	tt7972674	2
3425	tt7981260	4
3426	tt8011276	3
3427	tt8055888	1
3428	tt8060624	10
3429	tt8085616	7
3430	tt8108198	6
3431	tt8108202	4
3432	tt8136908	1
3433	tt8175968	2
3434	tt8202612	11
3435	tt8223250	9

	MID	female_cast
3436	tt8239946	3
3437	tt8324474	5
3438	tt8338746	5
3439	tt8396128	4
3440	tt8426854	6
3441	tt8427036	3
3442	tt8439854	5
3443	tt8458202	2
3444	tt8484590	3
3445	tt8581230	3
3446	tt8698956	5
3447	tt8852558	3
3448	tt8932884	1
3449	tt9007142	4

3450 rows × 2 columns

Now we calculate movies with only female cast by comparing total number of cast with above table

In [17]:
```
query = """
SELECT TRIM(MID), COUNT(TRIM(PID)) cast_count FROM M_Cast
WHERE TRIM(PID) IN (
  SELECT TRIM(PID) FROM Person
  WHERE Gender='Female'
)
GROUP BY MID

INTERSECT

SELECT TRIM(MID), COUNT(TRIM(PID)) cast_count FROM M_Cast
GROUP BY MID
"""
pd.read_sql_query(query, conn)
```

Out[17]:

	TRIM(MID)	cast_count
0	tt0272001	11
1	tt0354922	10
2	tt0375882	1
3	tt8458202	2

We get only 4 movies which have all female cast out of 3475 movies.

Now we build **FINAL QUERY**

In [18]:
```
query = f"""
SELECT {year_formula} real_year, COUNT(MID) no_of_female_cast_movies FROM Movie
WHERE MID IN (
SELECT MID FROM (
  SELECT MID, COUNT(TRIM(PID)) cast_count FROM M_Cast
  WHERE TRIM(PID) IN (
    SELECT TRIM(PID) FROM Person
    WHERE Gender='Female'
  )
  GROUP BY MID
  INTERSECT
  SELECT MID, COUNT(TRIM(PID)) cast_count FROM M_Cast
  GROUP BY MID
)
)
GROUP BY real_year
"""
pd.read_sql_query(query, conn)
```

Out[18]:

	real_year	no_of_female_cast_movies
0	1939	1
1	1999	1
2	2000	1
3	2018	1

I assume the data is not completely given in db file provided (db file size 7332 KB) so we get very sparse result. And we got only 4 rows and for other years the result is 0.

Question 5b:

We can biuld **FINAL QUERY** as we did most of the work in previous question.

In [19]:
```
query = f"""
SELECT table1.real_year, CAST(COALESCE(no_of_female_cast_movies, 0) AS FLOAT)*100/total_movies ratio, total_movi
FROM

(
  SELECT {year_formula} real_year, COUNT(MID) total_movies FROM Movie
  GROUP BY real_year
) table1

LEFT JOIN

(
  SELECT {year_formula} real_year, COUNT(MID) no_of_female_cast_movies FROM Movie
  WHERE MID IN (
    SELECT MID FROM (
      SELECT MID, COUNT(TRIM(PID)) cast_count FROM M_Cast
      WHERE TRIM(PID) IN (
        SELECT TRIM(PID) FROM Person
        WHERE Gender='Female'
      )
      GROUP BY MID

      INTERSECT

      SELECT MID, COUNT(TRIM(PID)) cast_count FROM M_Cast
      GROUP BY MID
    )
  )
  GROUP BY real_year
) table2

ON table1.real_year = table2.real_year
"""
pd.read_sql_query(query, conn)
```

Out[19]:

	real_year	ratio	total_movies
0	1931	0.000000	1
1	1936	0.000000	3
2	1939	50.000000	2

	real_year	ratio	total_movies
3	1941	0.000000	1
4	1943	0.000000	1
5	1946	0.000000	2
6	1947	0.000000	2
7	1948	0.000000	3
8	1949	0.000000	3
9	1950	0.000000	2
10	1951	0.000000	6
11	1952	0.000000	6
12	1953	0.000000	8
13	1954	0.000000	6
14	1955	0.000000	9
15	1956	0.000000	6
16	1957	0.000000	13
17	1958	0.000000	9
18	1959	0.000000	6
19	1960	0.000000	14
20	1961	0.000000	7
21	1962	0.000000	12
22	1963	0.000000	10
23	1964	0.000000	15
24	1965	0.000000	14
25	1966	0.000000	18
26	1967	0.000000	19
27	1968	0.000000	21
28	1969	0.000000	18
29	1970	0.000000	24

	real_year	ratio	total_movies
...	...	...	...
48	1989	0.000000	47
49	1990	0.000000	42
50	1991	0.000000	41
51	1992	0.000000	58
52	1993	0.000000	63
53	1994	0.000000	60
54	1995	0.000000	56
55	1996	0.000000	60
56	1997	0.000000	55
57	1998	0.000000	55
58	1999	1.515152	66
59	2000	1.562500	64
60	2001	0.000000	73
61	2002	0.000000	87
62	2003	0.000000	103
63	2004	0.000000	103
64	2005	0.000000	129
65	2006	0.000000	101
66	2007	0.000000	109
67	2008	0.000000	107
68	2009	0.000000	110
69	2010	0.000000	125
70	2011	0.000000	116
71	2012	0.000000	111
72	2013	0.000000	136
73	2014	0.000000	126

	real_year	ratio	total_movies
74	2015	0.000000	119
75	2016	0.000000	129
76	2017	0.000000	126
77	2018	0.961538	104

78 rows × 3 columns

We get 78 rows which has female_cast_movies ratio and total_movies count for all years

Question 6:

Seeing if more than one movie has the maximum cast value.

In [20]:
```
query = """
SELECT MID, COUNT(DISTINCT TRIM(PID)) cast_count FROM M_Cast
GROUP BY MID
ORDER BY cast_count DESC
LIMIT 10
"""
pd.read_sql_query(query, conn)
```

Out[20]:

	MID	cast_count
0	tt5164214	238
1	tt0451631	233
2	tt6173990	215
3	tt1188996	213
4	tt3498820	191
5	tt1981128	170
6	tt1573482	165
7	tt1190080	154
8	tt2120120	144
9	tt2510874	140

We get movie id with maximum cast from below query

In [21]:
```
query = """
SELECT MID, MAX(cast_count) FROM (
  SELECT MID, COUNT(DISTINCT TRIM(PID)) cast_count FROM M_Cast
  GROUP BY MID
)
"""
pd.read_sql_query(query, conn)
```

Out[21]:

	MID	MAX(cast_count)
0	tt5164214	238

FINAL QUERY

In [22]:
```
query = """
SELECT title, cast_count
FROM Movie m JOIN (
  SELECT MID, MAX(cast_count) cast_count FROM (
    SELECT MID, COUNT(DISTINCT TRIM(PID)) cast_count FROM M_Cast
    GROUP BY MID
  )
) id ON m.MID = id.MID
"""
pd.read_sql_query(query, conn)
```

Out[22]:

	title	cast_count
0	Ocean's Eight	238

"Ocean's Eight" is the movie with maximum cast of 238 distint people

Question 7:

Let us write query for getting all decades that are in our database.

In [23]:
```
query = f"""
SELECT DISTINCT {year_formula} start_year, {year_formula}+9 end_year
FROM Movie
ORDER BY start_year
"""
pd.read_sql_query(query, conn)
```

Out[23]:

	start_year	end_year
0	1931	1940
1	1936	1945
2	1939	1948
3	1941	1950
4	1943	1952
5	1946	1955
6	1947	1956
7	1948	1957
8	1949	1958
9	1950	1959
10	1951	1960
11	1952	1961
12	1953	1962
13	1954	1963
14	1955	1964
15	1956	1965
16	1957	1966
17	1958	1967
18	1959	1968
19	1960	1969
20	1961	1970
21	1962	1971

	start_year	end_year
22	1963	1972
23	1964	1973
24	1965	1974
25	1966	1975
26	1967	1976
27	1968	1977
28	1969	1978
29	1970	1979
...	...	...
48	1989	1998
49	1990	1999
50	1991	2000
51	1992	2001
52	1993	2002
53	1994	2003
54	1995	2004
55	1996	2005
56	1997	2006
57	1998	2007
58	1999	2008
59	2000	2009
60	2001	2010
61	2002	2011
62	2003	2012
63	2004	2013
64	2005	2014
65	2006	2015

	start_year	end_year
66	2007	2016
67	2008	2017
68	2009	2018
69	2010	2019
70	2011	2020
71	2012	2021
72	2013	2022
73	2014	2023
74	2015	2024
75	2016	2025
76	2017	2026
77	2018	2027

78 rows × 2 columns

Now join this decades with Movies Table on condition that movies year is in between the start_year and end_year. This will give number of movies in each decade.

In [24]:
```
query = f"""
SELECT decades.start_year, decades.end_year, COUNT(DISTINCT m.MID) no_of_movies
FROM (
  SELECT MID, {year_formula} real_year FROM Movie
) m JOIN (
  SELECT DISTINCT {year_formula} start_year, {year_formula}+9 end_year
  FROM Movie
  ORDER BY start_year
) decades ON m.real_year BETWEEN decades.start_year AND decades.end_year
GROUP BY decades.start_year
ORDER BY no_of_movies DESC
"""
pd.read_sql_query(query, conn)
```

Out[24]:

	start_year	end_year	no_of_movies
0	2008	2017	1205
1	2009	2018	1202
2	2007	2016	1188
3	2005	2014	1170
4	2006	2015	1160
5	2004	2013	1147
6	2003	2012	1114
7	2010	2019	1092
8	2002	2011	1090
9	2001	2010	1047
10	2000	2009	986
11	2011	2020	967
12	1999	2008	942
13	1998	2007	890
14	2012	2021	851
15	1997	2006	836
16	1996	2005	795

	start_year	end_year	no_of_movies
17	2013	2022	740
18	1995	2004	722
19	1994	2003	679
20	1993	2002	639
21	1992	2001	610
22	2014	2023	604
23	1991	2000	578
24	1990	1999	556
25	1989	1998	537
26	1988	1997	519
27	1987	1996	496
28	2015	2024	478
29	1986	1995	469
...	...	...	...
48	1968	1977	245
49	1967	1976	236
50	1966	1975	232
51	2017	2026	230
52	1965	1974	222
53	1964	1973	211
54	1963	1972	192
55	1962	1971	175
56	1961	1970	158
57	1960	1969	148
58	1959	1968	136
59	1958	1967	124
60	1957	1966	118

	start_year	end_year	no_of_movies
61	1956	1965	106
62	2018	2027	104
63	1955	1964	101
64	1954	1963	92
65	1953	1962	90
66	1952	1961	84
67	1951	1960	83
68	1950	1959	71
69	1949	1958	68
70	1948	1957	62
71	1947	1956	51
72	1946	1955	47
73	1943	1952	25
74	1941	1950	14
75	1939	1948	11
76	1936	1945	7
77	1931	1940	6

78 rows × 3 columns

Now we get the Maximum value from the above table. Building **FINAL QUERY** to get the answer. Ans notice that we dont have two decades with maximum number of movies from above result as the answer is sorted according to no_of_movies.

In [25]:
```
query = f"""
SELECT start_year, end_year, MAX(no_of_movies) FROM (
  SELECT decades.start_year, decades.end_year, COUNT(DISTINCT m.MID) no_of_movies
  FROM (
    SELECT MID, {year_formula} real_year FROM Movie
  ) m JOIN (
    SELECT DISTINCT {year_formula} start_year, {year_formula}+9 end_year
    FROM Movie
    ORDER BY start_year
  ) decades ON m.real_year BETWEEN decades.start_year AND decades.end_year
  GROUP BY decades.start_year
)
"""
pd.read_sql_query(query, conn)
```

Out[25]:

	start_year	end_year	MAX(no_of_movies)
0	2008	2017	1205

So The Decade [2008, 2017] has the maximum number of movies with 1205 movies made in between them.

Question 8:

src: [https://stackoverflow.com/questions/6299950/sql-difference-between-rows (https://stackoverflow.com/questions/6299950/sql-difference-between-rows)](https://stackoverflow.com/questions/6299950/sql-difference-between-rows)

Below code provides row number for each actor so actors having row number as only 1 did only one movie according to the dataset. So taking actors with atleast 2 movies and continue with the data.

In [26]:
```
query = f"""
WITH cte as (
  SELECT
    ROW_NUMBER() OVER (PARTITION BY actor_year.PID ORDER BY PID) row,
    PID, real_year
  FROM (
    SELECT DISTINCT TRIM(mc.PID) PID, {year_formula} real_year
    FROM Movie m JOIN M_Cast mc ON m.MID = mc.MID
    WHERE PID <> 'None'
    ORDER BY PID, real_year
  ) actor_year
)
SELECT * FROM cte
"""
pd.read_sql_query(query, conn)
```

Out[26]:

	row	PID	real_year
0	1	nm0000002	1959
1	1	nm0000027	1984
2	1	nm0000039	1953
3	1	nm0000042	1953
4	1	nm0000047	1970
5	1	nm0000073	1939
6	1	nm0000076	1977
7	1	nm0000092	2004
8	1	nm0000093	1997
9	1	nm0000096	2016
10	2	nm0000096	2017
11	1	nm0000101	2015
12	1	nm0000112	1988
13	1	nm0000113	2018
14	1	nm0000131	2009
15	1	nm0000137	2003

	row	PID	real_year
16	1	nm0000140	1996
17	1	nm0000144	2012
18	1	nm0000147	2008
19	1	nm0000155	2009
20	1	nm0000168	2012
21	1	nm0000173	2016
22	1	nm0000174	1996
23	1	nm0000187	2015
24	1	nm0000193	1996
25	2	nm0000193	2018
26	1	nm0000195	2007
27	1	nm0000200	2014
28	1	nm0000204	2007
29	1	nm0000207	2014
...	...	...	...
61957	1	nm9977802	2018
61958	1	nm9977803	2018
61959	1	nm9977805	2018
61960	1	nm9977806	2018
61961	1	nm9977807	2018
61962	1	nm9979161	2017
61963	1	nm9980716	2018
61964	1	nm9984753	2018
61965	1	nm9984754	2018
61966	1	nm9984755	2018
61967	1	nm9984756	2018
61968	1	nm9984757	2018

	row	PID	real_year
61969	1	nm9984758	2018
61970	1	nm9984759	2018
61971	1	nm9984760	2018
61972	1	nm9984761	2018
61973	1	nm9984763	2018
61974	1	nm9984764	2018
61975	1	nm9984765	2018
61976	1	nm9984766	2018
61977	1	nm9984767	2018
61978	1	nm9984768	2018
61979	1	nm9984769	2018
61980	1	nm9984770	2018
61981	1	nm9985086	2017
61982	1	nm9988016	2016
61983	1	nm9988018	2016
61984	1	nm9990703	2018
61985	1	nm9990704	2018
61986	1	nm9990705	2018

61987 rows × 3 columns

Same Code from above but now taking actors who acted in 2 movies or more.

In [27]:
```
query = f"""
WITH cte as (
  SELECT
    ROW_NUMBER() OVER (PARTITION BY actor_year.pid_t ORDER BY pid_t) row,
    pid_t, real_year
  FROM (
    SELECT DISTINCT TRIM(mc.PID) pid_t, {year_formula} real_year
    FROM Movie m JOIN M_Cast mc ON m.MID = mc.MID
    WHERE pid_t IN (
      SELECT pid_t FROM (
        SELECT TRIM(PID) pid_t, COUNT(DISTINCT MID) count FROM M_Cast
        GROUP BY pid_t
      ) WHERE count > 1
    )
    ORDER BY pid_t, real_year
  ) actor_year
)
SELECT * FROM cte
"""
pd.read_sql_query(query, conn)
```

Out[27]:

	row	pid_t	real_year
0	1	nm0000096	2016
1	2	nm0000096	2017
2	1	nm0000193	1996
3	2	nm0000193	2018
4	1	nm0000246	1996
5	2	nm0000246	2013
6	1	nm0000375	2005
7	2	nm0000375	2012
8	3	nm0000375	2016
9	1	nm0000673	2002
10	2	nm0000673	2016
11	1	nm0000818	1974

	row	pid_t	real_year
12	2	nm0000818	1975
13	3	nm0000818	1976
14	4	nm0000818	1977
15	5	nm0000818	1978
16	6	nm0000818	1979
17	7	nm0000818	1980
18	8	nm0000818	1981
19	9	nm0000818	1982
20	10	nm0000818	1983
21	11	nm0000818	1984
22	12	nm0000818	1986
23	13	nm0000818	1988
24	14	nm0000818	1989
25	15	nm0000818	1990
26	16	nm0000818	1991
27	17	nm0000818	1992
28	18	nm0000818	1994
29	19	nm0000818	1996
...	...	...	...
37569	2	nm9851522	2018
37570	1	nm9854347	2017
37571	1	nm9855558	2005
37572	2	nm9855558	2010
37573	3	nm9855558	2015
37574	4	nm9855558	2017
37575	1	nm9860260	2014
37576	2	nm9860260	2015

	row	pid_t	real_year
37577	1	nm9865122	2016
37578	2	nm9865122	2017
37579	1	nm9884041	2017
37580	2	nm9884041	2018
37581	1	nm9894731	2018
37582	1	nm9896474	2018
37583	1	nm9899526	2018
37584	1	nm9904342	2009
37585	2	nm9904342	2014
37586	3	nm9904342	2015
37587	1	nm9909965	2017
37588	1	nm9918523	2017
37589	2	nm9918523	2018
37590	1	nm9947307	2010
37591	2	nm9947307	2014
37592	1	nm9948998	2017
37593	2	nm9948998	2018
37594	1	nm9954913	2012
37595	2	nm9954913	2018
37596	1	nm9969854	1986
37597	2	nm9969854	2003
37598	3	nm9969854	2008

37599 rows × 3 columns

There is no definite range given in the original question so I am assuming the starting year of a actor is his/her career start and the last year he/she acted is his/her end of career.

If we consider like this and see the actor with id 'nm0000096' acted only in 2016 and 2017. So he is not unemployed more than 3 years considering his career started and ended at that years.

and if we consider actor with id 'nm9969854' he acted only in 1986, 2003 and 2008. So as he/she has gaps more than 3 years. So I dont consider this actor in my answer. (Both actor id's took are from start and end of the above table result).

By this assumption we may not get good results as lot of actors have only 2 movies which are not far apart year wise, and all those will be considered in our answer.

```
Now let us get the difference between the rows of same pid's
```

In [28]:
```
query = f"""
WITH cte as (
  SELECT
    ROW_NUMBER() OVER (PARTITION BY actor_year.pid_t ORDER BY pid_t) row,
    pid_t PID, real_year year
  FROM (
    SELECT DISTINCT TRIM(mc.PID) pid_t, {year_formula} real_year
    FROM Movie m JOIN M_Cast mc ON m.MID = mc.MID
    WHERE pid_t IN (
      SELECT pid_t FROM (
        SELECT TRIM(PID) pid_t, COUNT(DISTINCT MID) count FROM M_Cast
        GROUP BY pid_t
      ) WHERE count > 1
    )
    ORDER BY pid_t, real_year
  ) actor_year
)


SELECT a.PID, b.year - a.year
FROM cte a JOIN cte b ON a.PID = b.PID AND (a.row = b.row - 1)
"""
pd.read_sql_query(query, conn)
```

Out[28]:

	PID	b.year - a.year
0	nm0000096	1
1	nm0000193	22
2	nm0000246	17
3	nm0000375	7
4	nm0000375	4
5	nm0000673	14
6	nm0000818	1
7	nm0000818	1
8	nm0000818	1
9	nm0000818	1

	PID	b.year - a.year
10	nm0000818	1
11	nm0000818	1
12	nm0000818	1
13	nm0000818	1
14	nm0000818	1
15	nm0000818	1
16	nm0000818	2
17	nm0000818	2
18	nm0000818	1
19	nm0000818	1
20	nm0000818	1
21	nm0000818	1
22	nm0000818	2
23	nm0000818	2
24	nm0000818	1
25	nm0000818	1
26	nm0000818	1
27	nm0000818	1
28	nm0000818	2
29	nm0000818	1
...	...	...
29830	nm9709863	3
29831	nm9709863	1
29832	nm9709863	2
29833	nm9709863	3
29834	nm9709994	1
29835	nm9714216	4

	PID	b.year - a.year
29836	nm9734642	1
29837	nm9750158	1
29838	nm9764628	1
29839	nm9767719	1
29840	nm9767719	3
29841	nm9772472	9
29842	nm9784804	3
29843	nm9800848	2
29844	nm9831234	6
29845	nm9851522	5
29846	nm9855558	5
29847	nm9855558	5
29848	nm9855558	2
29849	nm9860260	1
29850	nm9865122	1
29851	nm9884041	1
29852	nm9904342	5
29853	nm9904342	1
29854	nm9918523	1
29855	nm9947307	4
29856	nm9948998	1
29857	nm9954913	6
29858	nm9969854	17
29859	nm9969854	5

29860 rows × 2 columns

As the above table seems correct we can continue to the **FINAL QUERY**

In [29]:
```
query = f"""
WITH cte as (
  SELECT
    ROW_NUMBER() OVER (PARTITION BY actor_year.pid_t ORDER BY pid_t) row,
    pid_t PID, real_year year
  FROM (
    SELECT DISTINCT TRIM(mc.PID) pid_t, {year_formula} real_year
    FROM Movie m JOIN M_Cast mc ON m.MID = mc.MID
    WHERE pid_t IN (
      SELECT pid_t FROM (
        SELECT TRIM(PID) pid_t, COUNT(DISTINCT MID) count FROM M_Cast
        GROUP BY pid_t
      ) WHERE count > 1
    )
    ORDER BY pid_t, real_year
  ) actor_year
)

SELECT DISTINCT TRIM(Name) FROM Person
WHERE TRIM(PID) IN (
  SELECT PID FROM (
    SELECT a.PID, (b.year - a.year) year_diff
    FROM cte a JOIN cte b ON a.PID = b.PID AND (a.row = b.row - 1)
  ) GROUP BY PID
  HAVING MAX(year_diff) <= 3
)
"""
pd.read_sql_query(query, conn)
```

Out[29]:

	TRIM(Name)
0	Steven Hauck
1	Serena Williams
2	Raj Awasti
3	Michael Chapman
4	James Heron
5	Alex Jaep
6	James Pimenta

	TRIM(Name)
7	Elena Valdameri
8	M'laah Kaur Singh
9	Stephen Bullard
10	Tina Grimm
11	Seth Zielicke
12	Sohum Shah
13	Piyush Kaushik
14	Rudra Soni
15	Sushant Singh Rajput
16	Alka Amin
17	Nishant Dahiya
18	Mir Sarwar
19	Martavious Gayles
20	Tahseen Ghauri
21	Jeff Glover
22	Inder Kumar
23	Duane Moseley
24	Patti Schellhaas
25	Manesh k Singh
26	Ayushmann Khurrana
27	Manav Vij
28	Ashwini Kalsekar
29	Chhaya Kadam
...	...
3060	Balbinder Dhami
3061	Kashi
3062	Vaijanath Biradar

	TRIM(Name)
3063	Ravi Srivatsa
3064	Gautam Joglekar
3065	Yagnesh Shetty
3066	Sanjay Amar
3067	Balaji Deshpande
3068	Nimish Gaur
3069	M.G. Soman
3070	Rajshree Sawant
3071	Nishan Nanaiah
3072	Gowri Pandit
3073	Satya Devi
3074	Lakhan Singh
3075	Romit Raaj
3076	Sripriya
3077	Qureishi
3078	Bhavya
3079	Ruzaan Bharucha
3080	Madhumeeta Das
3081	Roop Darshani
3082	Noor Naghmi
3083	Gaurav Bhattacharya
3084	Umashree
3085	Mamta Mohandas
3086	Kuldeep Ruhil
3087	Neha Julka
3088	Champa
3089	Vinu Chakravarthy

3090 rows × 1 columns

We got 3090 actors who are not unemployed for more than 3 years.

We got this high value because of the assumption I made, which is mentioned above. To get a satisfied answer there must be a range of years in which we should consider the careers of the actors. I assumed the career started with the starting year and ended with the last year. So the actors with only 2 movies nearby are all included in the answer. And I removed actors who only acted in one year so that our answer will less.

Question 9:

Getting count of movies for every actor and director. So the following output table have actor id and director id as unique to every row.

In [30]:
```
query = """
SELECT actor, director, no_of_movies FROM (
  SELECT TRIM(mc.PID) actor, TRIM(md.PID) director, COUNT(DISTINCT md.MID) no_of_movies
  FROM M_Director md JOIN M_Cast mc ON md.MID = mc.MID
  GROUP BY actor, director
)
ORDER BY no_of_movies DESC
"""
pd.read_sql_query(query, conn)
```

Out[30]:

	actor	director	no_of_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
10	nm0007147	nm0007147	13
11	nm0025630	nm0223522	13
12	nm0451864	nm0764316	13
13	nm0080324	nm0080315	12
14	nm0505323	nm0122216	12
15	nm0534501	nm0759662	12
16	nm0794510	nm0223522	12
17	nm1056425	nm0698184	12
18	nm0000821	nm0611531	11
19	nm0006795	nm0223522	11

	actor	director	no_of_movies
20	nm0707271	nm0007181	11
21	nm0848308	nm0223522	11
22	nm1259084	nm0698184	11
23	nm2136994	nm0223522	11
24	nm0004109	nm0080315	10
25	nm0006433	nm0223522	10
26	nm0007107	nm0890060	10
27	nm0007131	nm0007131	10
28	nm0222426	nm1460159	10
29	nm0451242	nm0223522	10
...	...	...	...
73380	nm9977802	nm3962028	1
73381	nm9977803	nm3962028	1
73382	nm9977805	nm3962028	1
73383	nm9977806	nm3962028	1
73384	nm9977807	nm3962028	1
73385	nm9979161	nm0151511	1
73386	nm9980716	nm9180204	1
73387	nm9984753	nm1150656	1
73388	nm9984754	nm1150656	1
73389	nm9984755	nm1150656	1
73390	nm9984756	nm1150656	1
73391	nm9984757	nm1150656	1
73392	nm9984758	nm1150656	1
73393	nm9984759	nm1150656	1
73394	nm9984760	nm1150656	1
73395	nm9984761	nm1150656	1

	actor	director	no_of_movies
73396	nm9984763	nm1150656	1
73397	nm9984764	nm1150656	1
73398	nm9984765	nm1150656	1
73399	nm9984766	nm1150656	1
73400	nm9984767	nm1150656	1
73401	nm9984768	nm1150656	1
73402	nm9984769	nm1150656	1
73403	nm9984770	nm1150656	1
73404	nm9985086	nm0151511	1
73405	nm9988016	nm1464314	1
73406	nm9988018	nm1464314	1
73407	nm9990703	nm4264671	1
73408	nm9990704	nm4264671	1
73409	nm9990705	nm4264671	1

73410 rows × 3 columns

Getting a table with actor as key and the director with he/she had done maximum number of movies. If the director id is equal to yash chopra, then we should take that actor in our answer but we also have to check if there is any other director who have same number of movies with same actor. So according to question the actors who have same maximum number of movies for more than one director shouldnt be allowed in the answer.

In [31]:
```
query = """
WITH count_of_movies AS (
  SELECT actor, director, no_of_movies FROM (
    SELECT TRIM(mc.PID) actor, TRIM(md.PID) director, COUNT(DISTINCT md.MID) no_of_movies
    FROM M_Director md JOIN M_Cast mc ON md.MID = mc.MID
    GROUP BY actor, director
  )
), max_count_movies AS (
  SELECT actor, director, no_of_movies FROM count_of_movies
  WHERE (actor, no_of_movies) IN (
    SELECT actor, MAX(no_of_movies) no_of_movies FROM count_of_movies
    GROUP BY actor
  )
)

SELECT actor, director, no_of_movies FROM max_count_movies
WHERE actor IN (
  SELECT actor FROM max_count_movies
  GROUP BY actor
  HAVING COUNT(director) = 1
)
ORDER BY no_of_movies DESC
"""
pd.read_sql_query(query, conn)
```

Out[31]:

	actor	director	no_of_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

	actor	director	no_of_movies
10	nm0007147	nm0007147	13
11	nm0025630	nm0223522	13
12	nm0451864	nm0764316	13
13	nm0080324	nm0080315	12
14	nm0505323	nm0122216	12
15	nm0534501	nm0759662	12
16	nm0794510	nm0223522	12
17	nm1056425	nm0698184	12
18	nm0000821	nm0611531	11
19	nm0006795	nm0223522	11
20	nm0707271	nm0007181	11
21	nm0848308	nm0223522	11
22	nm1259084	nm0698184	11
23	nm2136994	nm0223522	11
24	nm0004109	nm0080315	10
25	nm0006433	nm0223522	10
26	nm0007107	nm0890060	10
27	nm0007131	nm0007131	10
28	nm0222426	nm1460159	10
29	nm0451242	nm0223522	10
...	...	...	...
26096	nm9977802	nm3962028	1
26097	nm9977803	nm3962028	1
26098	nm9977805	nm3962028	1
26099	nm9977806	nm3962028	1
26100	nm9977807	nm3962028	1
26101	nm9979161	nm0151511	1

	actor	director	no_of_movies
26102	nm9980716	nm9180204	1
26103	nm9984753	nm1150656	1
26104	nm9984754	nm1150656	1
26105	nm9984755	nm1150656	1
26106	nm9984756	nm1150656	1
26107	nm9984757	nm1150656	1
26108	nm9984758	nm1150656	1
26109	nm9984759	nm1150656	1
26110	nm9984760	nm1150656	1
26111	nm9984761	nm1150656	1
26112	nm9984763	nm1150656	1
26113	nm9984764	nm1150656	1
26114	nm9984765	nm1150656	1
26115	nm9984766	nm1150656	1
26116	nm9984767	nm1150656	1
26117	nm9984768	nm1150656	1
26118	nm9984769	nm1150656	1
26119	nm9984770	nm1150656	1
26120	nm9985086	nm0151511	1
26121	nm9988016	nm1464314	1
26122	nm9988018	nm1464314	1
26123	nm9990703	nm4264671	1
26124	nm9990704	nm4264671	1
26125	nm9990705	nm4264671	1

26126 rows × 3 columns

If the director id in above table is "Yash Chopra's" then the corresponding actor should be in answer.

Now joining the table with M_Cast and M_Director to get our **FINAL ANSWER**.

In [34]:
```
query = """
WITH count_of_movies AS (
  SELECT actor, director, no_of_movies FROM (
    SELECT TRIM(mc.PID) actor, TRIM(md.PID) director, COUNT(DISTINCT md.MID) no_of_movies
    FROM M_Director md JOIN M_Cast mc ON md.MID = mc.MID
    GROUP BY actor, director
  )
), max_count_movies AS (
  SELECT actor, director, no_of_movies FROM count_of_movies
  WHERE (actor, no_of_movies) IN (
    SELECT actor, MAX(no_of_movies) no_of_movies FROM count_of_movies
    GROUP BY actor
  )
)

SELECT DISTINCT TRIM(Name) FROM Person
WHERE TRIM(PID) IN (
  SELECT actor FROM (
    SELECT actor, director, no_of_movies FROM max_count_movies
    WHERE actor IN (
      SELECT actor FROM max_count_movies
      GROUP BY actor
      HAVING COUNT(director) = 1
    )
  ) WHERE director IN (
    SELECT TRIM(PID) FROM Person
    WHERE Name = 'Yash Chopra'
  )
)
"""
pd.read_sql_query(query, conn)
```

Out[34]:

	TRIM(Name)
0	Waheeda Rehman
1	Achala Sachdev
2	Yash Chopra
3	Vinod Negi
4	Chandni Jas Keerat

	TRIM(Name)
5	Shivaya Singh
6	Huzefa Gadiwala
7	Manish Arora
8	Pankaj Raina
9	Neetu Singh
10	Julia St John
11	Susan Fordham
12	Steve Box
13	Jasmine Jardot
14	Abbie Murphy
15	Julie Vollono
16	Varun Thakur
17	Katy Kartwheel
18	Lucy Phelps
19	Matthew David McCarthy
20	Benjayx Murphy
21	Melissa Hollett
22	Nick Thomas-Webster
23	Anick Wiget
24	Jay Conroy
25	Gary Wronecki
26	Sean Moon
27	Rudy Valentino Grant
28	Richard Herdman
29	James Michael Rankin
...	...
75	Chandu Allahabadi

	TRIM(Name)
76	Shyam Arora
77	Leela Chitnis
78	Ravi Dubey
79	Ashok Verma
80	Pratima Puri
81	Ramola Bachchan
82	Sumati Gupte
83	Akhtar-Ul-Iman
84	Pran Mehra
85	Nissar
86	Master Rizwan
87	Naseem
88	Parijat
89	Nazir
90	Ram Maini
91	Om Sahni
92	Bhola
93	Ramanand
94	Gopal
95	Kishan
96	Nasir
97	Ashok Chadda
98	Rajesh
99	Master Kelly
100	Yasin Khan
101	Sandow S. Sethi
102	Naval

	TRIM(Name)
103	Prem Sood
104	Ramlal Shyamlal

105 rows × 1 columns

So we got 105 actors who made more movies with yash chopra than any other directors.

Question 10:

First calculating Shahrukh 1 actors.

In [35]:
```
query = """
SELECT DISTINCT TRIM(PID) FROM M_Cast
WHERE MID IN (
  SELECT MID FROM M_Cast
  WHERE TRIM(PID) IN (
    SELECT TRIM(PID) FROM Person WHERE TRIM(Name) = 'Shah Rukh Khan'
  )
)
"""
pd.read_sql_query(query, conn)
```

Out[35]:

	TRIM(PID)
0	nm0451321
1	nm0004418
2	nm1995953
3	nm2778261
4	nm0631373
5	nm0241935
6	nm0792116
7	nm1300111
8	nm0196375
9	nm1464837
10	nm2868019
11	nm0906226
12	nm1012529
13	nm0952232
14	nm0665555
15	nm2670109
16	nm2540053
17	nm0632634
18	nm2974366

	TRIM(PID)
19	nm3841022
20	nm3840343
21	nm0248474
22	nm2648101
23	nm2574505
24	nm2387689
25	nm4373080
26	nm2006282
27	nm2775421
28	nm2489172
29	nm2927987
...	...
2353	nm4968917
2354	nm0442470
2355	nm1267986
2356	nm0710601
2357	nm1236022
2358	nm0361643
2359	nm0759785
2360	nm0958860
2361	nm1942776
2362	nm2146462
2363	nm0474604
2364	nm0795348
2365	nm1886256
2366	nm1889672
2367	nm1894111

	TRIM(PID)
2368	nm2161830
2369	nm2145968
2370	nm0802369
2371	nm2135417
2372	nm0442479
2373	nm2133802
2374	nm2108016
2375	nm1543016
2376	nm2273524
2377	nm0080426
2378	nm4173451
2379	nm7620177
2380	nm3093045
2381	nm0451154
2382	nm3385526

2383 rows × 1 columns

So there are 2383 actors who acted with Shah Rukh Khan (Wow! :P). These are ShahRukh 1 actors. Now we have to calculate actors who are not in this list and are not 'Shah Rukh Khan' and acted with ShahRukh 1 actors. This next query is going to be nested a lot and going to be **FINAL QUERY**.

In [36]:
```
query = """
SELECT DISTINCT TRIM(Name) FROM Person
WHERE TRIM(PID) IN (
  SELECT DISTINCT TRIM(PID) FROM M_Cast
  WHERE MID IN (
    SELECT MID FROM M_Cast
    WHERE TRIM(PID) IN (
      SELECT DISTINCT TRIM(PID) FROM M_Cast
      WHERE MID IN (
        SELECT MID FROM M_Cast
        WHERE TRIM(PID) IN (
          SELECT TRIM(PID) FROM Person WHERE TRIM(Name) = 'Shah Rukh Khan'
        )
      )
    )
  ) AND TRIM(PID) NOT IN (
    SELECT DISTINCT TRIM(PID) FROM M_Cast
    WHERE MID IN (
      SELECT MID FROM M_Cast
      WHERE TRIM(PID) IN (
        SELECT TRIM(PID) FROM Person WHERE TRIM(Name) = 'Shah Rukh Khan'
      )
    )
  ) AND TRIM(PID) NOT IN (
    SELECT TRIM(PID) FROM Person WHERE TRIM(Name) = 'Shah Rukh Khan'
  )
)
"""
pd.read_sql_query(query, conn)
```

Out[36]:

	TRIM(Name)
0	Freida Pinto
1	Rohan Chand
2	Damian Young
3	Waris Ahluwalia
4	Caroline Christl Long
5	Rajeev Pahuja
6	Michelle Santiago

	TRIM(Name)
7	Alicia Vikander
8	Dominic West
9	Walton Goggins
10	Daniel Wu
11	Kristin Scott Thomas
12	Derek Jacobi
13	Alexandre Willaume
14	Tamer Burjaq
15	Adrian Collins
16	Keenan Arrison
17	Andrian Mazive
18	Milton Schorr
19	Hannah John-Kamen
20	Peter Waison
21	Samuel Mak
22	Sky Yang
23	Civic Chung
24	Josef Altin
25	Billy Postlethwaite
26	Roger Jean Nsengiyumva
27	Jaime Winstone
28	Michael Obiora
29	Shekhar Varma
...	...
24278	Surjeet Redi
24279	Premji
24280	Kamu

	TRIM(Name)
24281	Monal
24282	Kanchan Mullick
24283	Sumit Ganguly
24284	Aindrila Banerjee
24285	Aritro Dutta Banik
24286	Soham Chakraborty
24287	Hiran Chatterjee
24288	Mallika Ghosh
24289	Subrata Guharoy
24290	Raju Majumdar
24291	Sudipta Majumdar
24292	Vishwesh Krishnamoorthy
24293	Sabitha Perara
24294	R. Sundarajan
24295	Ushma Rathod
24296	Zubeda Khan
24297	N. Sagar
24298	Habib Tanvar
24299	Mohd. Zahiruddin
24300	Muktha George
24301	Anjuman
24302	Dhruv Shetty
24303	Minoo Mehtab
24304	Hayley Cleghorn
24305	Nirvasha Jithoo
24306	Kamal Maharshi
24307	Mohini Manik

24308 rows × 1 columns

There are 24308 actors who are ShahRukh 2 actors. And the ShahRukh 1 actors are not included in this. And Shah Rukh Khan is also not included in this list.

Default Template for query

In [ ]:
```
query = """

"""
pd.read_sql_query(query, conn)
```