Backslash

Python Project

Avirup Roy Chowdhury

Pranathi Varma M.

Niharika Gupta

Chaitanya Mohite

Diptajit Chaurangi

Problem Statement

To design a python application which will be able to find common columns between multiple tables of a database without using SQL joins and retrieve data, this data will later be written to a file using IO operations.

In this case we'll be using a sample database called chinook for the operations, to better demonstrate the implementation we will be searching the entire database for a column called **PlaylistID** without manually finding out where the column exists.

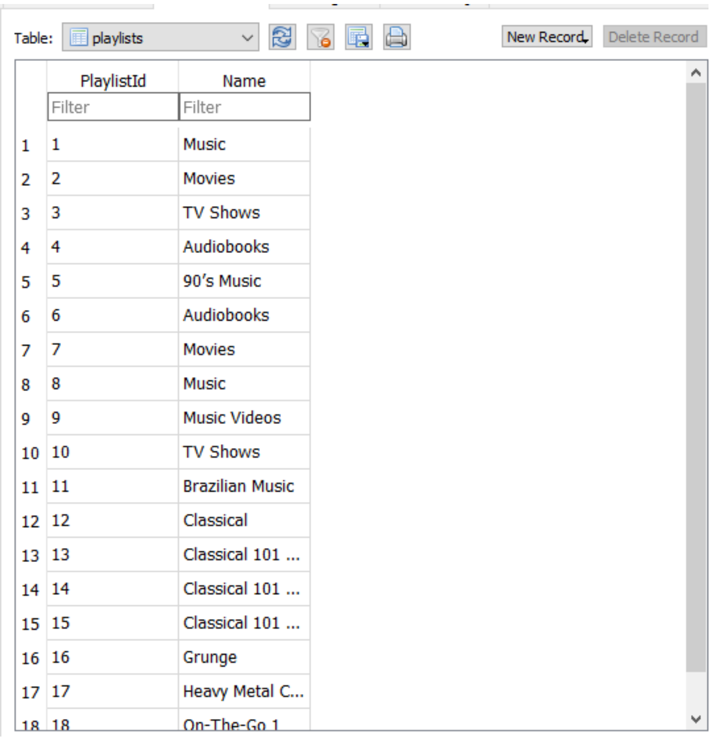
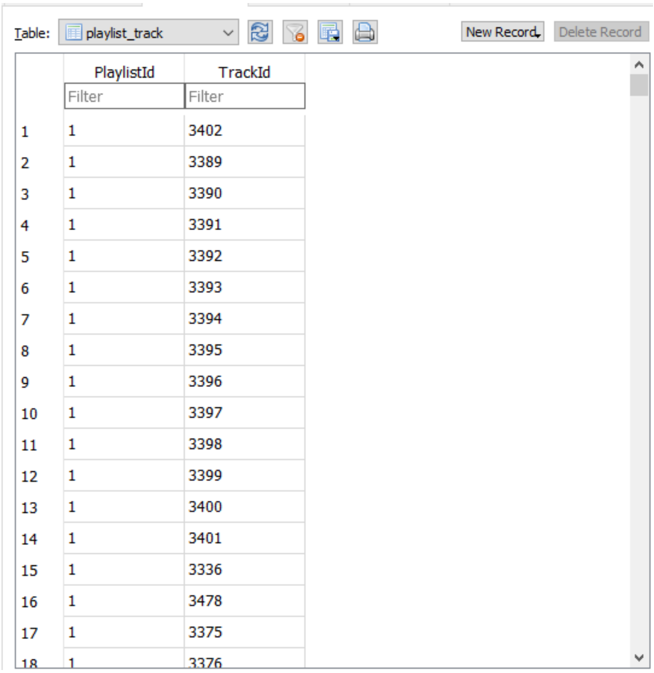


Table: playlists

	PlaylistId	Name
1	1	Music
2	2	Movies
3	3	TV Shows
4	4	Audiobooks
5	5	90's Music
6	6	Audiobooks
7	7	Movies
8	8	Music
9	9	Music Videos
10	10	TV Shows
11	11	Brazilian Music
12	12	Classical
13	13	Classical 101 ...
14	14	Classical 101 ...
15	15	Classical 101 ...
16	16	Grunge
17	17	Heavy Metal C...
18	18	On-The-Go 1

Fig 1 : PlaylistID exists in *playlists*

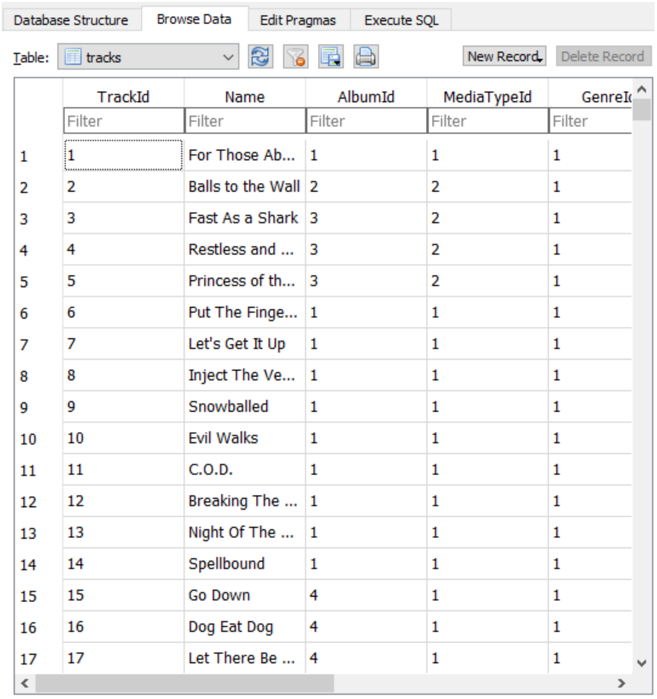
Once **PlaylistID** is found which should exist in a table called *playlists* we will take up 4 data members and find their corresponding **TrackID** in a table called *playlist_track.*



	PlaylistId	TrackId
	Filter	Filter
1	1	3402
2	1	3389
3	1	3390
4	1	3391
5	1	3392
6	1	3393
7	1	3394
8	1	3395
9	1	3396
10	1	3397
11	1	3398
12	1	3399
13	1	3400
14	1	3401
15	1	3336
16	1	3478
17	1	3375
18	1	3376

Fig 2 : TrackID exists in *playlist_tracks*

The **TrackID** is further referenced to another table called *tracks* from where **Name** and **AlbumID** will be then further exported to a file called *export.txt.*



	TrackId	Name	AlbumId	MediaTypeId	GenreI
	Filter	Filter	Filter	Filter	Filter
1	1	For Those Ab...	1	1	1
2	2	Balls to the Wall	2	2	1
3	3	Fast As a Shark	3	2	1
4	4	Restless and ...	3	2	1
5	5	Princess of th...	3	2	1
6	6	Put The Finge...	1	1	1
7	7	Let's Get It Up	1	1	1
8	8	Inject The Ve...	1	1	1
9	9	Snowballed	1	1	1
10	10	Evil Walks	1	1	1
11	11	C.O.D.	1	1	1
12	12	Breaking The ...	1	1	1
13	13	Night Of The ...	1	1	1
14	14	Spellbound	1	1	1
15	15	Go Down	4	1	1
16	16	Dog Eat Dog	4	1	1
17	17	Let There Be ...	4	1	1

Fig 3 : Comparison Table

Technologies Used

The project base and structure will be revolving around **Python 3** and will be implemented using two libraries i.e

1. **Pandas -** an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming
2. **Sqlite3** - a relational database management system contained in a C library. In contrast to many other database management systems, SQLite is not a client–server database engine. Rather, it is embedded into the end program.

Scope of the Project

The project will revolve around finding a general column name in multiple tables of a database and find relations among different columns name related to first and finally printing the data in a file.

Source Code and Algorithm

1) **Import the required library**

```
import sqlite3
import pandas as pd
con = sqlite3.connect('chinook.db')
cur = con.cursor()
```

2) **Adding all database to a list**

```
a=['albums','artists','playlists','playlist_track','tracks']
d={}
```

3) **Getting column names and table names in which these columns are present in a dictionary**

```
for val in a:
    print(val)
    cur.execute("select * from {}".format(val))
    col_list=cur.description
l1=[]
for i in range(len(col_list)):
    l1.append(col_list[i][0])
```

```
print(l1)
for col in l1:
    if col in d:
     d[col].append(val)
     else:
     d[col]=[val]
```

4) Fetching table name in which columns ‘trackID’ and ‘playlistID’ both are present into a list

```
l1=d['TrackId']
l2=d['PlaylistId']
print(l1)
print(l2)
l3=list(set(l1)&set(l2))
print(l3)
```

5) Creating a dictionary for TrackID and their corresponding 10 track ids

```
for i in range(len(playlist)):
cur.execute("select * from {} where PlaylistId={}".format(l3[0],playlist[i]))
list_c=cur.fetchall()
print(list_c)
for j in range(10):
  if list_c[j][0] in dict_of_tracks:
  dict_of_tracks[list_c[j][0]].append(list_c[j][1])
  else:
  dict_of_tracks[list_c[j][0]]=[list_c[j][1]]
```

6) Fetching table name in which column names ‘trackID’ and ‘albumID’ are both present in a list

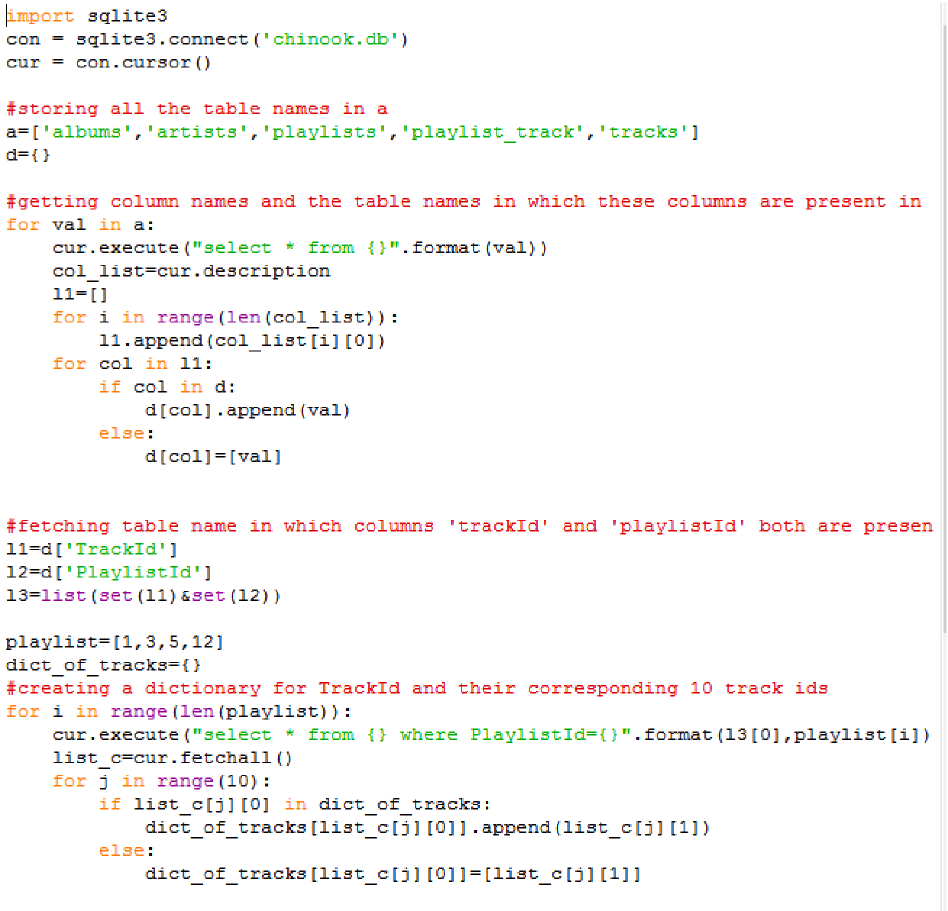
```
tableb=list(set(l1)&set(d['AlbumId']))
```

7) Storing it into a file

```
f=open('f_out_final.txt','a')
f.write("trackId \t")
f.write("albumid \t")
f.write("name \t")
f.write("\n")
for i in range(len(dict_of_tracks)):
       for j in range(len(dict_of_tracks[playlist[i]])):
       cur.execute("select TrackId,AlbumId,Name from {} where TrackId={}".format(tableb[0],dict_of_tracks[playlist[i]][j]))
       listd=cur.fetchall()
```

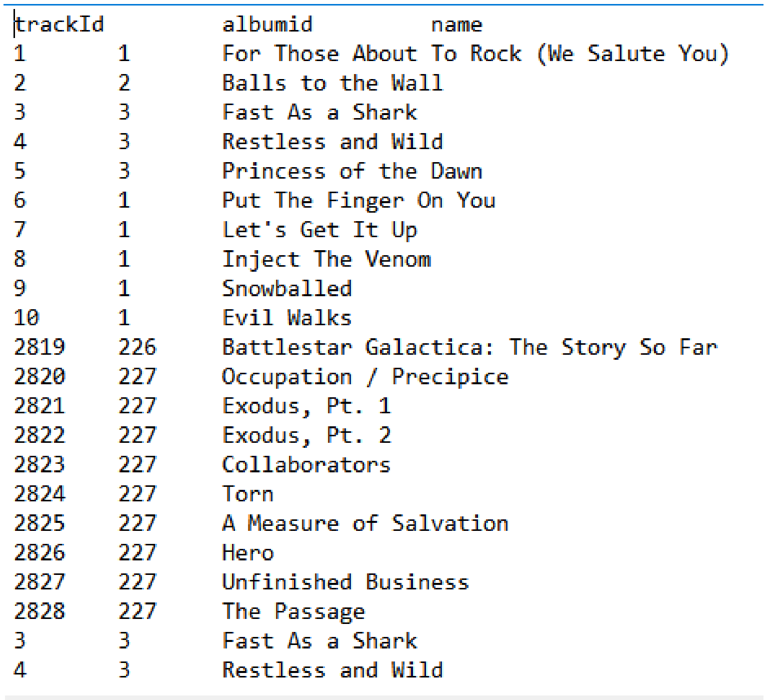
```
print(listd)  f.write(str(listd[0][0]) +'\t')
f.write(str(listd[0][1]) +'\t') f.write(listd[0][2])
f.write("\n")
f.close()
```

Screenshots



```
import sqlite3
con = sqlite3.connect('chinook.db')
cur = con.cursor()

#storing all the table names in a
a=['albums','artists','playlists','playlist_track','tracks']
d={}

#getting column names and the table names in which these columns are present in
for val in a:
    cur.execute("select * from {}".format(val))
    col_list=cur.description
    l1=[]
    for i in range(len(col_list)):
        l1.append(col_list[i][0])
    for col in l1:
        if col in d:
            d[col].append(val)
        else:
            d[col]=[val]


#fetching table name in which columns 'trackId' and 'playlistId' both are presen
l1=d['TrackId']
l2=d['PlaylistId']
l3=list(set(l1)&set(l2))

playlist=[1,3,5,12]
dict_of_tracks={}
#creating a dictionary for TrackId and their corresponding 10 track ids
for i in range(len(playlist)):
    cur.execute("select * from {} where PlaylistId={}".format(l3[0],playlist[i])
    list_c=cur.fetchall()
    for j in range(10):
        if list_c[j][0] in dict_of_tracks:
            dict_of_tracks[list_c[j][0]].append(list_c[j][1])
        else:
            dict_of_tracks[list_c[j][0]]=[list_c[j][1]]
```

Fig 4 : CODE



```
trackId         albumid         name
1       1       For Those About To Rock (We Salute You)
2       2       Balls to the Wall
3       3       Fast As a Shark
4       3       Restless and Wild
5       3       Princess of the Dawn
6       1       Put The Finger On You
7       1       Let's Get It Up
8       1       Inject The Venom
9       1       Snowballed
10      1       Evil Walks
2819    226     Battlestar Galactica: The Story So Far
2820    227     Occupation / Precipice
2821    227     Exodus, Pt. 1
2822    227     Exodus, Pt. 2
2823    227     Collaborators
2824    227     Torn
2825    227     A Measure of Salvation
2826    227     Hero
2827    227     Unfinished Business
2828    227     The Passage
3       3       Fast As a Shark
4       3       Restless and Wild
```

Fig 5 : output

Result

The final output has been achieved using all the specified constraints and specifications i.e without execution of SQL queries and the results have been verified to work.

Libraries and Functions Used:

1. SQLite3

- ***a.*** ***Connect() –*** *a connection objet to connect to database*
- *b.* ***Cursor() –*** *object to call execute method to execute queries*
- ***c.*** ***Execute() –*** *to perform SQL commands*
- ***d.*** ***Fetchall() –*** *to get a list of matching rows*
- ***e.*** ***Close() –*** *to close the database*
- ***f.*** ***Commit() –*** *to save the changes permanently in the database*