DJANGO UNCHAINED

By

Siddarth Naik

Sai Balaguru Ganess

V. Jahnavi Reddy

Swetha Vemparala

# INTRODUCTION:

The aim of this project is to traverse through a list of tables which are interlinked through a primary key and then print the columns required from the last table. Using the attributes of first table, which are not primary keys, we search for a table which has the non-key attributes of first table as the primary key of second table. Here, finding this new table is a task since we are also running a piece of code to find its location.

### SQLITE3:

SQLITE is a [relational database management system](https://en.wikipedia.org/wiki/Relational_database_management_system) (RDBMS) contained in a [C](https://en.wikipedia.org/wiki/C_(programming_language)) [library](https://en.wikipedia.org/wiki/Library_(computer_science)). In contrast to many other database management systems, SQLite is not a [client–server](https://en.wikipedia.org/wiki/Client%E2%80%93server) database engine. Rather, it is embedded into the end program.

SQLite is [ACID](https://en.wikipedia.org/wiki/ACID)-compliant and implements most of the [SQL](https://en.wikipedia.org/wiki/SQL) standard, generally following [PostgreSQL](https://en.wikipedia.org/wiki/PostgreSQL) syntax. However, SQLite uses a dynamically and weakly [typed](https://en.wikipedia.org/wiki/Data_type) SQL [syntax](https://en.wikipedia.org/wiki/Syntax) that does not guarantee the [domain integrity](https://en.wikipedia.org/wiki/Data_integrity#TYPES). This means that one can, for example, insert a string into a column defined as an integer. SQLite will attempt to convert data between formats where appropriate, the string "123" into an integer in this case, but does not guarantee such conversions, and will store the data as-is if such a conversion is not possible.

SQLite is a popular choice as [embedded database](https://en.wikipedia.org/wiki/Embedded_database) software for local/client storage in [application software](https://en.wikipedia.org/wiki/Application_software) such as [web browsers](https://en.wikipedia.org/wiki/Web_browser). It is arguably the most widely deployed [database engine](https://en.wikipedia.org/wiki/Database_engine), as it is used today by several widespread browsers, [operating systems](https://en.wikipedia.org/wiki/Operating_system), and [embedded systems](https://en.wikipedia.org/wiki/Embedded_system) (such as mobile phones), among others. SQLite has [bindings](https://en.wikipedia.org/wiki/Language_binding) to many programming languages.

## SOURCE CODE:

import pandas as pd

import sqlite3

Here we are importing pandas and SQLITE3.

con = sqlite3.connect("chinook.db")

cur = con.cursor()

query = """ Select name from sqlite\_master where type = "table" """

Here we are establishing the connection with the database and selecting the table names present in the database and also creating a cursor to get the data.

res = cur.execute(query)

list\_tables = res.fetchall()

Here we are executing the query and fetching the data that is in the cursor and putting all the table names into the list.

list\_tables

Viewing the list.

table\_name\_columns = {}

for t\_name in list\_tables:

q1 = 'SELECT \* FROM {}'.format(str(t\_name[0]))

cursor = con.execute(q1)

temp1 = []

for i in range(len(cursor.description)):

temp1.append(cursor.description[i][0])

table\_name\_columns[str(t\_name[0])] = temp1

Here we are creating a dictionary and inserting the column names of each table into the dictionary.

table\_name\_columns

Viewing the dictionary.

name\_col = table\_name\_columns['playlists'][0]

for key in table\_name\_columns.keys():

if name\_col in table\_name\_columns[key]:

new\_table = key

Here we are searching for the non-key attributes of the table and searching for the table which has this non-key attribute as the primary key. Thereby, we are creating a new table and inserting this searched attribute in it as a primary key.

new\_table

Viewing the table.

q2 = "select \* from {}".format(new\_table)

cur2 = con.execute(q2)

playlist\_track = cur2.fetchall()

Here we are fetching the new\_table values.

id = [1,3,5,8]

track\_id = {}

for j in id:

counter = 0

temp = []

for i in range(len(playlist\_track)):

if playlist\_track[i][0] == j:

if(counter!=10):

counter+=1

temp.append(playlist\_track[i][1])

else:

break

track\_id[j] = temp

Here we are creating a dictionary in which the keys are taken from the playlist and each playlist has ten track\_ids which are the values and are kept in a list.

track\_id

Viewing the track\_id dictionary.

name\_col1 = table\_name\_columns['playlist\_track'][1]

for key in table\_name\_columns.keys():

if name\_col1 in table\_name\_columns[key] and key != 'playlist\_track':

new\_table1 = key

We are creating a new table which is going to accommodate the non-key attribute of previous table as the primary key.

new\_table1

Viewing the table.

q3 = "select \* from {}".format(new\_table1)

cur3 = con.execute(q3)

tracks = cur3.fetchall()

tracks

Here we are fetching all the values from tracks and viewing.

track\_name = {}

for id in track\_id.keys():

for i in track\_id[id]:

for j in range(len(tracks)):

if tracks[j][0] == i:

track\_name[i]=[tracks[j][1],tracks[j][2]]

Creating a dictionary that has track\_id as the key and track\_name, album\_id as the values.

track\_name

Viewing the dictionary.

name\_col2 = table\_name\_columns['tracks'][2]

**for** key **in** table\_name\_columns.keys():

**if** name\_col2 **in** table\_name\_columns[key] **and** key != 'tracks':

new\_table2 = key

Here we are finding the album\_id by traversing from tracks to the album table.

new\_table2

Viewing albums table.

q4 = "select \* from {}".format(new\_table2)

cur4 = con.execute(q4)

albums = cur4.fetchall()

To fetch the contents of table albums

albums

viewing albums

for id in track\_name.keys():

for j in range(len(albums)):

if albums[j][0] == track\_name[id][1]:

track\_name[id].append(albums[j][1])

track\_name[id].append(albums[j][2])

With album\_id, we are fetching album\_name and artist\_id

track\_name

Viewing track\_name

name\_col3 = table\_name\_columns['albums'][2]

for key in table\_name\_columns.keys():

if name\_col3 in table\_name\_columns[key] and key != 'albums':

new\_table3 = key

Getting artist\_name from artist\_id

new\_table3

q5 = "select \* from {}".format(new\_table3)

cur5 = con.execute(q5)

artists = cur5.fetchall()

Creating a new table and fetching all

artists

Viewing contents of artists

for id in track\_name.keys():

for j in range(len(artists)):

if artists[j][0] == track\_name[id][3]:

track\_name[id].append(artists[j][1])

Appending the contents of tracks to fetched values from artists

track\_name

Viewing final contents of track\_name

df = pd.DataFrame()

Creating a new dataframe

df['track\_id'] = track\_name.keys()

song\_name = []

album\_id = []

album\_name = []

artist\_id = []

artist\_name = []

Creating lists for all columns

for key in track\_name.keys():

song\_name.append(track\_name[key][0])

album\_id.append(track\_name[key][1])

album\_name.append(track\_name[key][2])

artist\_id.append(track\_name[key][3])

artist\_name.append(track\_name[key][4])

Appending the values accordingly into lists

df['Song title'] = song\_name

df['Album\_id'] = album\_id

df['Album Name'] = album\_name

df['Artist ID'] = artist\_id

df['Artist Name'] = artist\_name

Inserting the lists into dataframes

play\_id = []

Creating new list

x = list(df['track\_id'])

for i in x:

if i in track\_id[1]:

play\_id.append(1)

elif i in track\_id[3]:

play\_id.append(3)

elif i in track\_id[5]:

play\_id.append(5)

elif i in track\_id[8]:

play\_id.append(8)

Inserting values into play\_id.

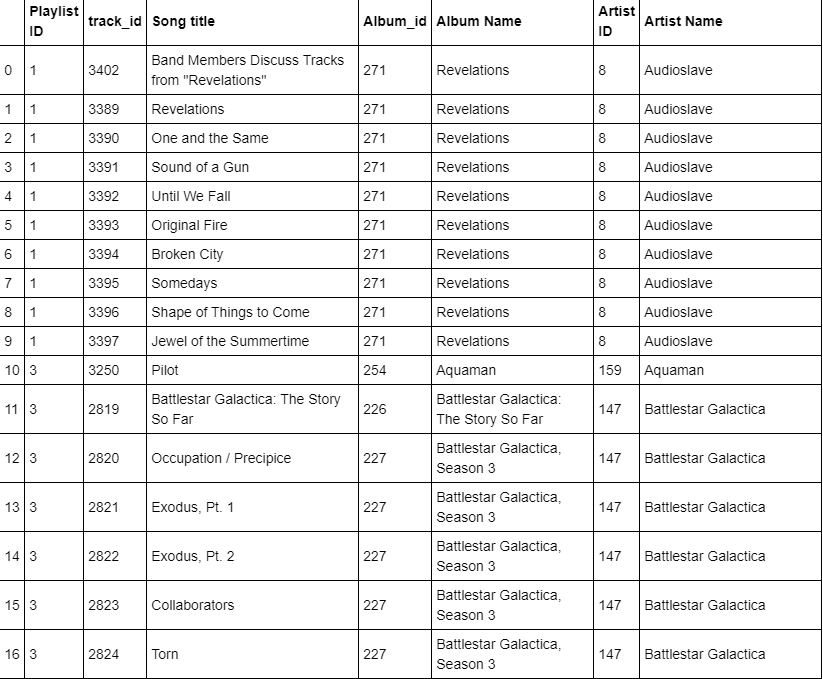
play\_id

Viewing contents of play\_id.

df.insert(0,'Playlist ID',play\_id)

df

Viewing contents of data frame.



# Conclusion:

The given task has been completed and executed successfully.