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 packages.

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 in “chinook.db” into “list\_tables”

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 “table\_name\_columns” and inserting the column names of each table into the dictionary with “Table Name” as Keys.

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

We search for the table which has a column named “playlist\_id” and it’s stored in the variable new\_table. The table is “playlist\_tracks”

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

cur2 = con.execute(q2)

playlist\_track = cur2.fetchall()

Here we are fetching the track\_id from playlist\_track table.

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

We have created list named “ID” and hard-coded the values as [1,3,5,8]. We fetch the respective “track\_id” for each Playlist\_id and store it in a dictionary – “track\_id”

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 search for another table which contains the column – “Track\_id” and store it in new\_table1. The Table is: “Tracks”

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

cur3 = con.execute(q3)

tracks = cur3.fetchall()

Here we are fetching all the values from table “Tracks”.

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 the list of track\_name, album\_id as the values.

.

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 search for another table that contains the column “album\_id” and it’s stored in new\_table2. The table is: “albums”

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

cur4 = con.execute(q4)

albums = cur4.fetchall()

To fetch the contents of table 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 and it’s appended to the values of respective “track\_id” in the dictionary “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

We traverse to the “artist” table using “Artist\_id”. The table name is stored in new\_table3.

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

cur5 = con.execute(q5)

artists = cur5.fetchall()

We store the data in the list “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 artist\_name from artists table to the values 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])

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

Appending the list for each column and concatenating it to the Data frame.

Inserting the lists into dataframes

play\_id = []

Creating new list for the playlist\_id.

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)

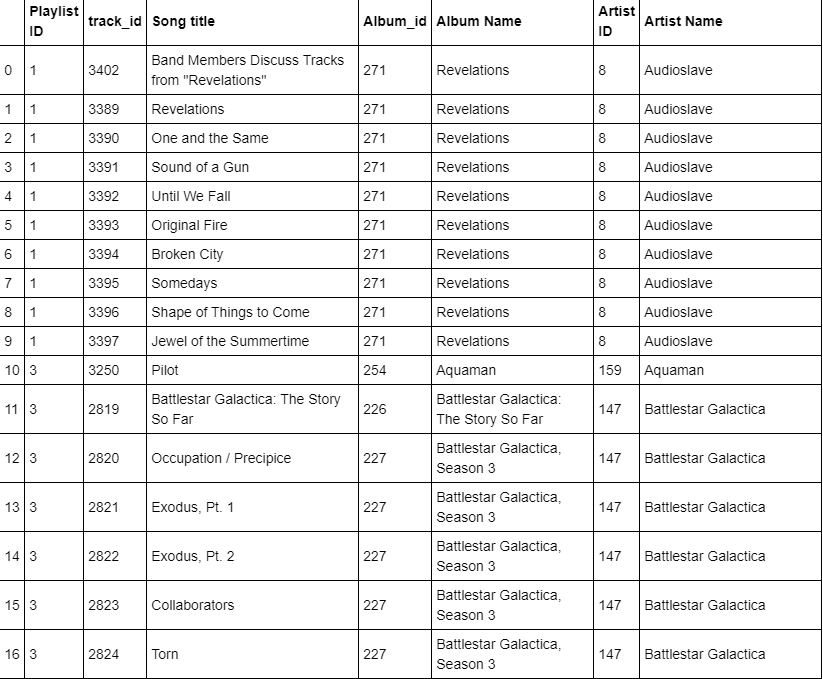
Inserting values into the list play\_id in accordance with the Track\_id.

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

Inserting the column “Playlist\_id” to the Data Frame “df”

df

Viewing contents of data frame.



# Conclusion:

The given task has been completed and executed successfully.