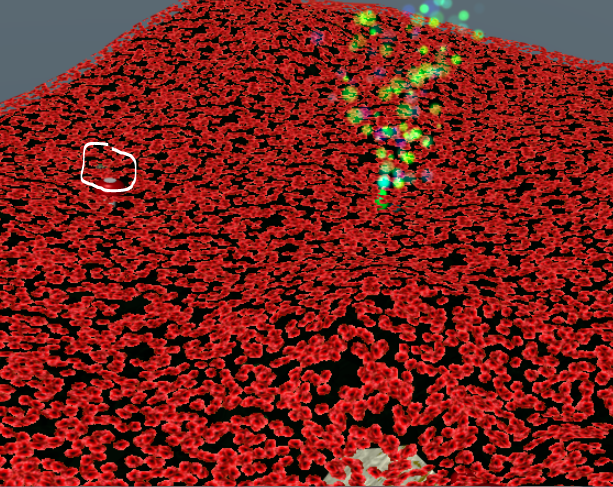
# I Don’t Think We’re in Kansas Linux Challenge or Christmas Songs Data Analysis Terminal Challenge

The terminal is in the “I Don’t Think We’re in Kanas Any More” game, but it’s hard to find. It’s under the “poppy” floor. I’ve put a bumper near it to make it easier to find. If you use the SVGS login, you can find the terminal on the Stocking page as Christmas Songs Data Analysis Terminal Challenge.



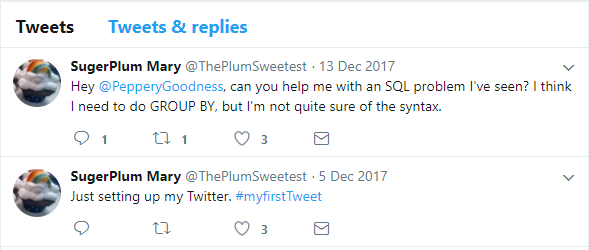
The object is to find the most popular song in the Christmas Songs database, christmassongs.db. The database is a SQLite database, which is an open source database that is included in the C library and is very popular.

Basic SQL skills are useful for cyber security practitioners, and for IT professionals in general. This challenge will provide practice on basic SQL commands.

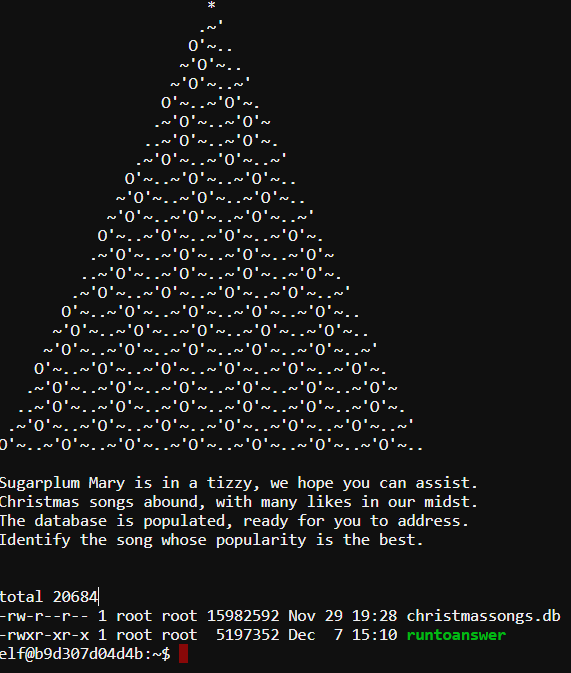
There is a good Pen Test Blog article to help you. <https://pen-testing.sans.org/blog/2017/12/09/your-pokemon-guide-for-essential-sql-pen-test-commands> Note: If you want to practice with SQL, and SQLite in particular, you can download the Pokemon database they use here, <http://veekun.com/static/pokedex/downloads/veekun-pokedex.sqlite.gz>.

## Hint

SugarPlum Mary has sent a tweet that says she needs to use GROUP BY, which will help us. We’ll need to use the GROUP BY command as explained in the blog, at some point.

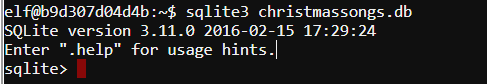


The opening screen shows us the database file. Once we have the answer to Mary’s question, we’ll execute ./runtoanswer to see if we got it right.



## Method

To access the database, run  
sqlite3 christmassongs.db



Now follow the steps listed in the blog article:

1. Enumerate the tables. Use the .schema command to list the tables. In the article, the table names “item\_pockets” and “pokeathalon\_stats” are shown. 1) What are the tables in the Christmas Songs database?
2. Enumerate the fields in the tables. Each table has several fields that store data. In the article, the fields of the table pokemon are identifier, species\_id, height, weight, base\_experience, order, and is\_default. Key fields are unique, so that they uniquely identify one record as being different from all other records. For example, each Pokémon character has its own identifier in the pokemon table, even if other fields (species, height, weight) may be the same as other characters. Also note that the species\_id field references (should match) the id field of the pokemon\_species table. That way, if you know the identifier of a Pokémon character, you can find additional information about its species by looking up the species\_id in the id field of the species table. 2) What are the fields of the tables in the Christmas Songs database? What is the key of each table? Do any fields in one table reference fields in other tables--if so, what are they?
3. Examine some data. Practice using the SELECT statement as shown in the “Retrieving Some or All” portion of the article. Some notes:
   1. The database is big so be sure to limit your queries to just a few responses. For example, select \* from <tablename> LIMIT 5; would be a good idea
   2. The commands are not case sensitive. Select and SELECT, limit and LIMIT, etc., do the same thing.
   3. Each command must end with a semicolon. If the terminal shows you a …> prompt it is expecting more input. Most likely, you forgot the semicolon. When your commands get long, you can use this to your advantage to split a long command into multiple lines.
   4. 3) Show the results of a simple query (screenshot, or cut/paste)
4. Use Conditional Expressions. Our job is to determine the most popular song by counting the number of likes it has. The table “likes” has a field called “like” that we will use. We need to know what the possible values of the like field are.
   1. Write a query (select \* from likes where <???> limit 5;) to see if the like field can be < 1.
   2. Write a query to see if the like field can be = 1 or > 1.
   3. 4) What are the possible values of the like field?
5. Skip ahead to “Aggregate Functions” in the article. We need to count the number of likes for each song, so the COUNT function may be helpful.
   1. Count the total number of records in the like table using count(songid)
   2. Now add a where clause that counts records with the like field equal to 1, and not 0. Your statement should look something like the pokemon statement below but adjusted for our database.    
      5) How many records do you get?
   3. Name your count so we can refer to it later, i.e.  
      count(songid) as some\_cool\_name and run your query again.
6. The previous step gave us the total number of likes but didn’t tell us how many likes went with each song. To do that, we’ll have to use the GROUP BY statement. Each song has a unique songid, so songid should be a good field to group by. Add songid to the list of fields you select (i.e. count(songid), songid). Then add, group by songid limit 15, to the end of your query. That should show a column of counts and songids. The “limit 15” is there because there are a lot of songs.
7. The previous step is close, but it would be nice to have the list in order by count, so we can easily tell which song has the most counts. Read the “Ordering Data” section of the article. Remember that we gave a name to our count column in step 5c? Add, order by some\_cool\_name [asc or desc], to the end of your query. Use either asc for ascending, or desc for descending, as appropriate. 6) Which songid has the most likes?
8. We need the song title, not the songid. You can run a simple query, select title from songs where id = <songid you came up with> to find the title. 7) What is the song title with the most likes?
9. The previous step works, but it isn’t the coolest way. What if your job was to come up with a list of the Top 10 most popular songs? You would have to run the previous step 10 times and then collect the results. Read the section “Cross Table Queries” and adjust your query so that it lists the Top 10 songs. 8) Show a screenshot of your query and the Top 10 songs.

## Questions

1. Submit the answers to questions 1-8, above.