


# This is CS50

CS50's Introduction to Computer Science

OpenCourseWare

Donate  (<https://cs50.harvard.edu/donate>)

David J. Malan (<https://cs.harvard.edu/malan/>)

[malan@harvard.edu](mailto:malan@harvard.edu)

 (<https://www.facebook.com/dmalan>)  (<https://github.com/dmalan>) 

(<https://www.instagram.com/davidjmalan/>)  ([https://www.linkedin.com](https://www.linkedin.com/in/malan/)

[in/malan/](https://www.linkedin.com/in/malan/))  (<https://www.reddit.com/user/davidjmalan>) 

(<https://www.threads.net/@davidjmalan>)  (<https://twitter.com/davidjmalan>)

## Songs



## Problem to Solve

Write SQL queries to answer questions about a database of the 100 most-streamed songs on [Spotify](https://en.wikipedia.org/wiki/Spotify) (<https://en.wikipedia.org/wiki/Spotify>) in 2018.

## Demo

```
$ sqlite3 songs.db
SQLite version 3.40.1 2022-12-28 14:03:47
Enter ".help" for usage hints.
sqlite>
```

Recorded with [asciinema](#)

## Getting Started

For this problem, you'll use a database provided to you by CS50's staff.

### ▼ Download the distribution code

Open [VS Code \(https://cs50.dev/\)](https://cs50.dev/).

Start by clicking inside your terminal window, then execute `cd` by itself. You should find that its “prompt” resembles the below.

```
$
```

Click inside of that terminal window and then execute

```
wget https://cdn.cs50.net/2023/fall/psets/7/songs.zip
```

followed by Enter in order to download a ZIP called `songs.zip` in your codespace. Take care not to overlook the space between `wget` and the following URL, or any other character for that matter!

Now execute

```
unzip songs.zip
```

to create a folder called `songs`. You no longer need the ZIP file, so you can execute

```
rm songs.zip
```

and respond with “y” followed by Enter at the prompt to remove the ZIP file you downloaded.

Now type

```
cd songs
```

followed by Enter to move yourself into (i.e., open) that directory. Your prompt should now resemble the below.

```
songs/ $
```

If all was successful, you should execute

```
ls
```

and you should see 8 .sql files, `songs.db`, and `answers.txt`.

If you run into any trouble, follow these same steps again and see if you can determine where you went wrong!

## Understanding

Provided to you is a file called `songs.db`, a SQLite database that stores data from [Spotify](https://developer.spotify.com/documentation/web-api/) (<https://developer.spotify.com/documentation/web-api/>) about songs and their artists. This dataset contains the top 100 streamed songs on Spotify in 2018. In a terminal window, run `sqlite3 songs.db` so that you can begin executing queries on the database.

First, when `sqlite3` prompts you to provide a query, type `.schema` and press enter. This will output the `CREATE TABLE` statements that were used to generate each of the tables in the database. By examining those statements, you can identify the columns present in each table.

Notice that every `artist` has an `id` and a `name`. Notice, too, that every song has a `name`, an `artist_id` (corresponding to the `id` of the artist of the song), as well as values for the danceability, energy, key, loudness, speechiness (presence of spoken words in a track), valence, tempo, and duration of the song (measured in milliseconds).

The challenge ahead of you is to write SQL queries to answer a variety of different questions by selecting data from one or more of these tables. After you do so, you'll reflect on the ways Spotify might use this same data in their annual [Spotify Wrapped](https://en.wikipedia.org/wiki/Spotify_Wrapped) ([https://en.wikipedia.org/wiki/Spotify\\_Wrapped](https://en.wikipedia.org/wiki/Spotify_Wrapped)) campaign to characterize listeners' habits.

## Implementation Details

---

For each of the following problems, you should write a single SQL query that outputs the results specified by each problem. Your response must take the form of a single SQL query, though you may nest other queries inside of your query. You **should not** assume anything about the `id`s of any particular songs or artists: your queries should be accurate even if the `id` of any particular song or person were different. Finally, each query should return only the data necessary to answer the question: if the problem only asks you to output the names of songs, for example, then your query should not also output each song's tempo.

1. In `1.sql`, write a SQL query to list the names of all songs in the database.
  - Your query should output a table with a single column for the name of each song.
2. In `2.sql`, write a SQL query to list the names of all songs in increasing order of tempo.
  - Your query should output a table with a single column for the name of each song.
3. In `3.sql`, write a SQL query to list the names of the top 5 longest songs, in descending order of length.
  - Your query should output a table with a single column for the name of each song.
4. In `4.sql`, write a SQL query that lists the names of any songs that have danceability, energy, and valence greater than 0.75.
  - Your query should output a table with a single column for the name of each song.
5. In `5.sql`, write a SQL query that returns the average energy of all the songs.
  - Your query should output a table with a single column and a single row containing the average energy.
6. In `6.sql`, write a SQL query that lists the names of songs that are by Post Malone.
  - Your query should output a table with a single column for the name of each song.
  - You should not make any assumptions about what Post Malone's `artist_id` is.
7. In `7.sql`, write a SQL query that returns the average energy of songs that are by Drake.
  - Your query should output a table with a single column and a single row containing the average energy.
  - You should not make any assumptions about what Drake's `artist_id` is.
8. In `8.sql`, write a SQL query that lists the names of the songs that feature other artists.

- Songs that feature other artists will include “feat.” in the name of the song.
- Your query should output a table with a single column for the name of each song.

## Hints

See [this SQL keywords reference \(https://www.w3schools.com/sql/sql\\_ref\\_keywords.asp\)](https://www.w3schools.com/sql/sql_ref_keywords.asp) for some SQL syntax that may be helpful!

### ▼ List the names of all songs in the database

Recall that, to select all values in a table's column, you can use SQL's `SELECT` keyword. `SELECT` is followed by the column (or columns) you'd like to select, which is in turn followed by `FROM table` where `table` is the name of the table you'd like to select from.

In `1.sql`, then, try writing the following:

```
-- All songs in the database.  
SELECT name  
FROM songs;
```

### ▼ List the names of all songs in increasing order of tempo

Recall that SQL has an `ORDER BY` keyword, by which you can order the results of your query by the value in a certain column. For example, `ORDER BY tempo` will order results by the column `tempo`.

In `2.sql`, then, try writing the following:

```
-- All songs in increasing order of tempo.  
SELECT name  
FROM songs  
ORDER BY tempo;
```

### ▼ List the names of the top 5 longest songs, in descending order of length

Recall that `ORDER BY` need not always sort in ascending order. You can specify that your results be sorted in *descending* order by appending `DESC`. For example, `ORDER BY duration_ms DESC` will list results in descending order, by duration.

And recall, too, that `LIMIT n` can specify that you only want the first  $n$  rows that match a specific query. For example, `LIMIT 5` will return only the first five results of the query.

In `3.sql`, then, try writing the following:

```
-- The names of the top 5 longest songs, in descending order of length.
```

```
SELECT name
FROM songs
ORDER BY duration_ms DESC
LIMIT 5;
```

#### ▼ List the names of any songs that have danceability, energy, and valence greater than 0.75

Recall that you can filter results in SQL with `WHERE` clauses, which are followed by some condition which typically tests the values in a row's columns.

Recall, too, that SQL's operators function much the same way as C's. For example, `>` evaluates to "true" when the value on the left is greater than the value on the right. You may chain these expressions together, using `AND` or `OR`, to form one larger condition.

In `4.sql`, then, try writing the following:

```
-- The names of any songs that have danceability, energy, and valence greater than 0.75
SELECT name
FROM songs
WHERE danceability > 0.75 AND energy > 0.75 AND valence > 0.75;
```

#### ▼ Find average energy of all the songs

Recall that SQL supports keywords not just to select particular rows, but also to *aggregate* the data in those rows. In particular, you might find the `AVG` keyword (to compute averages) useful. To aggregate the results of a column, just apply the aggregation function to that column. For example, `SELECT AVG(energy)` will find the average of the values in the energy column for the given query.

In `5.sql`, then, try writing the following:

```
-- The average energy of all the songs.
SELECT AVG(energy)
FROM songs;
```

#### ▼ List the names of songs that are by Post Malone

Notice that, if you execute `.schema songs` in your sqlite prompt, the `songs` table has song names but not artist names! Instead, `songs` has an `artist_id` column. To list the names of songs by Post Malone, then, you'll first need to identify Post Malone's artist id.

```
-- Identify Post Malone's artist id
SELECT id
FROM artists
WHERE name = 'Post Malone';
```

This query returns 54. Now, you could query the `songs` table for any song with Post Malone's id.

```
SELECT name
FROM songs
WHERE artist_id = 54;
```

But, per the specification, you should be mindful not to assume knowledge of any ids. You could improve the design of this query by *nesting* your two queries.

In `6.sql`, then, try writing the following:

```
-- The names of songs that are by Post Malone.
SELECT name
FROM songs
WHERE artist_id =
(
    SELECT id
    FROM artists
    WHERE name = 'Post Malone'
);
```

#### ▼ Find the average energy of songs that are by Drake

Notice that, similar to the previous query, you'll need to combine multiple tables to successfully run this query. You could again use nested subqueries, but consider another approach too!

Recall that you can use SQL's `JOIN` keyword to combine multiple tables into one, so long as you specify which columns across those tables should ultimately match. For example, the following query joins the `songs` and `artists` tables, indicating that the `artist_id` column in the `songs` table and the `id` column in the `artists` table should match:

```
SELECT *
FROM songs
JOIN artists ON songs.artist_id = artists.id
```

With these two tables combined, it's only a matter of filtering your selection to find the average energy of songs by Drake.

In `7.sql`, then, try writing the following:

```
-- The average energy of songs that are by Drake
SELECT AVG(energy)
FROM songs
JOIN artists ON songs.artist_id = artists.id
WHERE artists.name = 'Drake';
```

#### ▼ List the names of the songs that feature other artists

For this query, note that songs which feature other artists typically have some mention of "feat." in their title. Recall that SQL's `LIKE` keyword can be used to match strings with

certain phrases (like “feat.”!). To do so, you can use `%`: a wildcard character that matches any sequence of characters.

In `8.sql`, then, try writing the following:

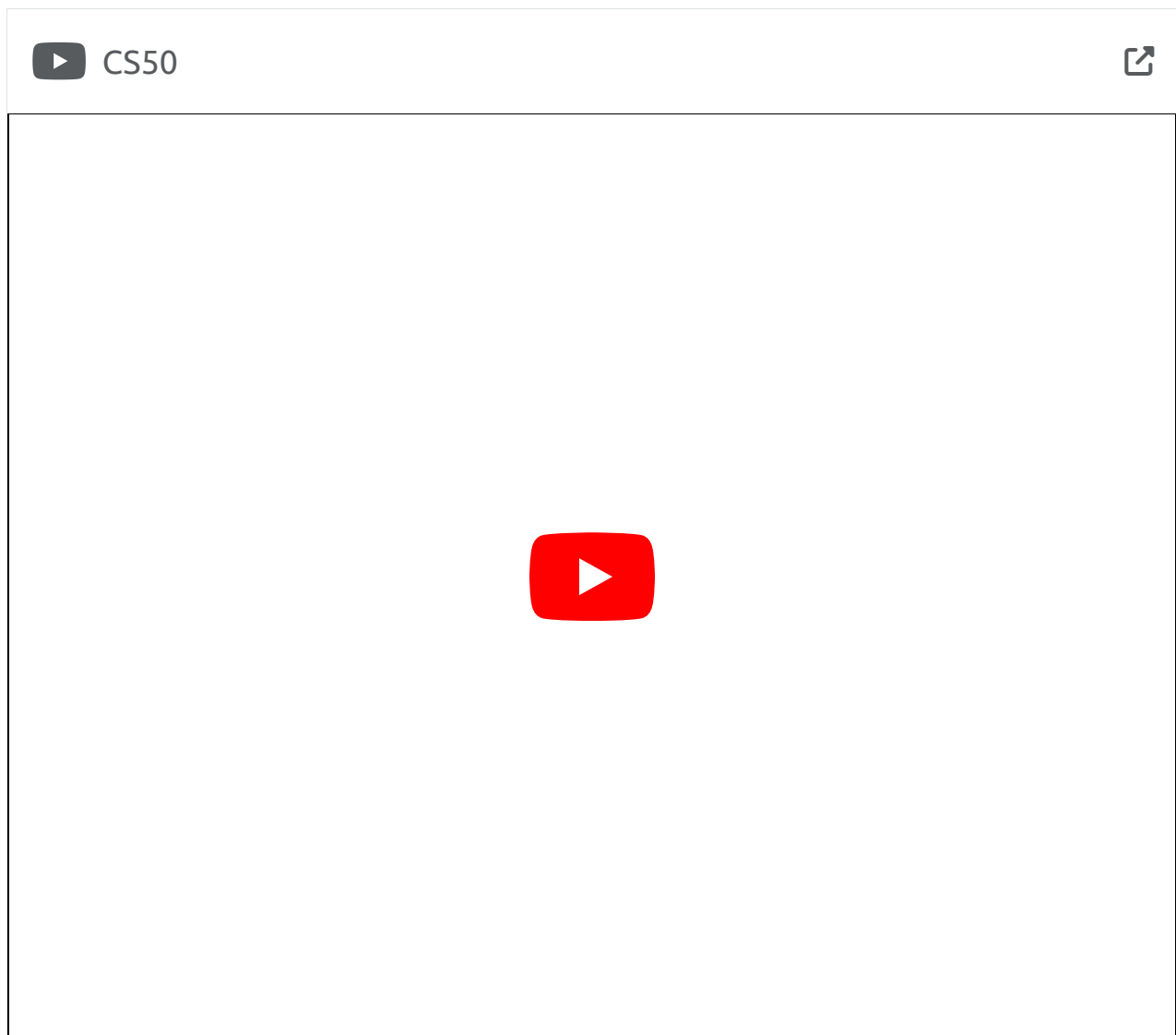
```
-- The names of songs that feature other artists.  
SELECT name  
FROM songs  
WHERE name LIKE '%feat.%';
```

## Walkthrough



▼ Not sure how to solve?





## Spotify Wrapped

Spotify Wrapped ([https://en.wikipedia.org/wiki/Spotify\\_Wrapped](https://en.wikipedia.org/wiki/Spotify_Wrapped)) is a feature presenting Spotify users' 100 most played songs from the past year. In 2021, Spotify Wrapped calculated an "Audio Aura" (<https://newsroom.spotify.com/2021-12-01/learn-more-about-the-audio-aura-in-your-spotify-2021-wrapped-with-aura-reader-mystic-michaela/>) for each user, a "reading of [their] two most prominent moods as dictated by [their] top songs and artists of the year." Suppose Spotify determines an audio aura by looking at the average energy, valence, and danceability of a person's top 100 songs from the past year. In `answers.txt`, reflect on the following questions:

- If `songs.db` contains the top 100 songs of one listener from 2018, how would you characterize their audio aura?
- Hypothesize about why the way you've calculated this aura might *not* be very representative of the listener. What better ways of calculating this aura would you propose?

Be sure to submit `answers.txt` along with each of your `.sql` files!

## How to Test

---

### Correctness

```
check50 cs50/problems/2024/x/songs
```

## How to Submit

---

```
submit50 cs50/problems/2024/x/songs
```

## Acknowledgements

---

Dataset from [Kaggle \(https://www.kaggle.com/nadintamer/top-spotify-tracks-of-2018\)](https://www.kaggle.com/nadintamer/top-spotify-tracks-of-2018).