CS50's Introduction to Databases with SQL

OpenCourseWare

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

Carter Zenke (https://carterzenke.me)
carter@cs50.harvard.edu
(https://github.com/carterzenke) (https://www.linkedin.com/in/carterzenke/)

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan)
(https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/)
(https://www.reddit.com/user/davidjmalan) (https://www.threads.net/@davidjmalan)
(https://twitter.com/davidjmalan)

The Private Eye

"A detective's library in the style of an oil painting," generated by DALL·E 2 (https://openai.com/dall-e-2)

Problem to Solve

CS50's duck debugger has disappeared once more and you desperately need a detective. You've heard stories of one who lives uptown, always secretive in their work, never seen but when they

want to be. Unsurprisingly, they've proven to be quite elusive to you. But here you are, in their study, after picking up their address from a certain mail clerk.

On their mahogany desk, fresh white paper glints. Inscribed is the following table:

14	98	4
114	3	5
618	72	9
630	7	3
932	12	5
2230	50	7
2346	44	10
3041	14	5

And tucked underneath, a worn book, *The Adventures of Sherlock Holmes* (https://en.wikipedia.org/wiki/The_Adventures_of_Sherlock_Holmes).

Distribution Code

For this problem, you'll need to download `private.db` and `private.sql`.

▶ **Download the distribution code**

Background

Given the paper's proximity to *The Adventures of Sherlock Holmes*, what's written on it seems to be some variation of a book cipher (https://en.wikipedia.org/wiki/Book_cipher). You know that in one version of a book cipher, the cipher's creator gives you a list of "triplets" (i.e., a set of three numbers). Each triplet is structured as follows:

- The first number in the triplet is the *sentence number* referenced by the encoder.
- The second number in the triplet is the *character number*, within that sentence, at which the message begins.
- The third number in the triplet is the *message length* in characters (i.e., how many characters to read from the first, including spaces and punctuation).

For instance, consider the triplet **2**, **1**, **8** in light of the following sentences:

> Quite so! You have not observed. And yet you have seen.

2 refers to the 2nd sentence, "You have not observed." **1** refers to the 1st character in that sentence, "Y." And **8** refers to the length of the message from that first character. Starting from the 1st character of the 2nd sentence, reading 8 characters (including spaces!) gives you:

> You have

You can imagine, now, stringing together multiple tuples to encode a longer message. Perhaps that's exactly what the detective has done!

Schema

Within `private.db`, you'll find a table, `sentences`. The `sentences` table contains all sentences in *The Adventures of Sherlock Holmes*. In particular, it contains the following columns:

- `id`, which is the ID of the sentence
- `sentence`, which is the sentence itself

Specification

Your task at hand is to decode the cipher left for you by the detective. How you do so is up to you, but you should ensure that—at the end of your process—you have a **view** structured as follows:

- The view should be named `message`
- The view should have a single column, `phrase`
- When the following SQL query is executed on `private.db`, your view should return a single column in which each row is one phrase in the message.

```
SELECT "phrase" FROM "message";
```

In `private.sql`, you should write *all* SQL statements required to replicate your creation of the view. That is:

- If creating the view requires creating a separate table and inserting data into it, you should ensure that `private.sql` contains the statements to create that table and insert that data. (Don't be afraid to add tables and add data as you wish!)
- `private.sql`, when run a fresh instance of `private.db`, should be able to fully reconstruct your view.

Advice

Turns out that SQLite handily comes with a function that implements the very functionality of the book cipher: `substr` (https://www.sqlitetutorial.net/sqlite-functions/sqlite-substr/). The function `substr` takes three inputs ("arguments"):

- A string (i.e., text) from which to take a substring (i.e., a subset of the string's characters)
- The starting character of the substring, identified by its number (the first character is 1)
- The length of the substring

For instance, suppose you have a table called `sentences` which includes the following:

id **sentence**	**1** Quite so!
id **sentence**	**2** You have not observed.
id **sentence**	**3** And yet you have seen.

Should you run the following SQL query…

```
SELECT substr("sentence", 1, 2)
FROM "sentences";
```

you would receive the following results:

substr("sentence", 1, 2)	**Qu**
substr("sentence", 1, 2)	**Yo**
substr("sentence", 1, 2)	**An**

Keep in mind that the other arguments to `substr` can be entire columns themselves, too!

How to Test

Correctness

Execute the below to evaluate the correctness of your findings using `check50`

```
check50 cs50/problems/2023/sql/private
```

How to Submit

In your terminal, execute the below to submit your work.

```
submit50 cs50/problems/2023/sql/private
```

Acknowledgements

Text retrieved *The Adventures of Sherlock Holmes*, part of the public domain.