Meteorite Cleaning

"NASA astronauts cleaning a big meteorite, in the style of a science-fiction comic", generated by DALL·E 2

Problem to Solve

As a data engineer at NASA, you often spend your time cleaning meteorites—or at least the data they create.

You've been given a CSV file of historical meteorite landings here on Earth, of which there are quite a few! Your job is to import the data into a SQLite database, cleaning it up along the way. After you're done, the database will be used in analyses by some of your fellow engineers.

Distribution Code

For this problem, you'll need to download `meteorites.csv`, along with an `import.sql` file in which you'll write SQL statements to clean the CSV.

▼ **Download the distribution code**

Log into cs50.dev, click on your terminal window, and execute `cd` by itself. You should find that your terminal window's prompt resembles the below:

```
$
```

Next execute

```
wget https://cdn.cs50.net/sql/2024/x/psets/3/meteorites.zip
```

in order to download a ZIP called `meteorites.zip` into your codespace.

Then execute

```
unzip meteorites.zip
```

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

```
rm meteorites.zip
```

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

Now type

```
cd meteorites
```

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

```
meteorites/ $
```

If all was successful, you should execute

```
ls
```

and see a CSV file named `meteorites.csv` alongside an `import.sql` file. If not, retrace your steps and see if you can determine where you went wrong!

Specification

In `import.sql`, write a series of SQL (and SQLite) statements to import and clean the data from `meteorites.csv` into a table, `meteorites`, in a database called `meteorites.db`.

Within `meteorites.db`, the `meteorites` table should have the following columns:

▼ **Columns in the `meteorites` table**

- `id`, which represents the unique ID of the meteorite.
- `name`, which represents the given name of the meteorite.
- `class`, which is the classification of the meteorite, according to the traditional classification scheme.
- `mass`, which is the weight of the meteorite, in grams.
- `discovery`, which is either **"Fell"** or **"Found"**. "Fell" indicates the meteorite was seen falling to Earth, whereas "Found" indicates the meteorite was found only *after* landing on Earth.
- `year`, which is the year in which the the meteorite was discovered.
- `lat`, which is the latitude at which the meteorite landed.
- `long`, which is the longitude at which the meteorite landed.

Keep in mind that not all columns in the CSV should end up in the final table!

To consider the data in the `meteorites` table clean, you should ensure…

1. Any empty values in `meteorites.csv` are represented by `NULL` in the `meteorites` table.
    - Keep in mind that the `mass`, `year`, `lat`, and `long` columns have empty values in the CSV.
2. All columns with decimal values (e.g., 70.4777) should be rounded to the nearest hundredths place (e.g., 70.4777 becomes 70.48).
    - Keep in mind that the `mass`, `lat`, and `long` columns have decimal values.
3. All meteorites with the `nametype` "Relict" are *not* included in the `meteorites` table.
4. The meteorites are sorted by `year`, oldest to newest, and then—if any two meteorites landed in the same year—by `name`, in alphabetical order.
5. You've updated the IDs of the meteorites from `meteorites.csv`, according to the order specified in #4.
    - The `id` of the meteorites should start at 1, beginning with the meteorite that landed in the oldest year and is the first in alphabetical order for that year.

Advice

It can feel overwhelming to know where to start when cleaning such a big data file! Let's break the problem down into smaller pieces.

▼ **Begin by importing `meteorites.csv` into a temporary table**

Start by getting all of the data from `meteorites.csv` into a temporary table, one called `meteorites_temp`. A temporary table is a helpful placeholder: you can use it to clean your data until it's in a form that's suitable for your final `meteorites` table.

Before you import a CSV into a SQLite database, it's best to define the schema for the table into which that data will be imported. In `import.sql`, then, try the following:

```
CREATE TABLE "meteorites_temp" (
    -- TODO
);
```

We'll leave the column names up to you.

Next, recall that `.import` is a SQLite statement that can import a CSV into a table of your choice. After your `CREATE TABLE` statement, write a `.import` statement to import the data from `meteorites.csv` into the `meteorites_temp` table.

Finally, per the Usage section below, try creating `meteorites.db` by running the statements in `import.sql`.

▼ **Write SQL statements to clean the imported data**

With your data in a temporary table, continue writing SQL statements to clean the data. Consider how you might update the values of the `mass` column for instance:

```
UPDATE "meteorites_temp"
SET "mass" = ...
WHERE ...
```

You might need to write a few such statements, one (or more) for each column you're trying to clean.

▼ **Transfer the data from your temporary table into a `meteorites` table**

Recall that you can `INSERT` values into a new table by `SELECT`ing rows from another:

```
INSERT INTO "table0" ("column0", "column1")
SELECT "column0", "column1" FROM "table1";
```

When you do so, you can re-order your data using `ORDER BY`. And, so long as you've specified a primary key column in your new table, such a statement will auto-assign new IDs to the inserted rows if none is specified.

Once you're done with the temporary table, it's good practice to drop it!

Usage

Let's introduce a few terminal commands that might come in handy while you're working on cleaning this data set! Consider the following:

```
cat import.sql | sqlite3 meteorites.db
```

The above command can be broken down into two parts:

- `cat import.sql` outputs the data in `import.sql`. Try it by itself if you're curious.
- `sqlite3 meteorites.db` opens a file called `meteorites.db` with the `sqlite3` engine, as you're already familiar with.

When these commands are combined with a pipe, `|`, the data from `import.sql` is treated as a set of statements for `sqlite3` to run on `meteorites.db`. If `meteorites.db` doesn't yet exist, it will be created and the statements in `import.sql` will be run on it.

What if your `import.sql` isn't perfect and you want to re-create the database? Consider deleting the current version of `meteorites.db` with:

```
rm meteorites.db
```

`rm` stands for remove. If prompted, type "y" followed by enter to confirm the deletion of `meteorites.db`. From there, you can re-run `cat import.sql | sqlite3 meteorites.db` to create `meteorites.db` from scratch.

Looking to speed things up? You can use the up arrow to access previously typed commands in your terminal.

How to Test

While `check50` is available for this problem, you're encouraged to also test your code on your own. Try using the commands in Usage to see your current progress towards a clean data set!

Correctness

```
check50 cs50/problems/2024/sql/meteorites
```

How to Submit

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

```
submit50 cs50/problems/2024/sql/meteorites
```

Acknowledgements

Meteorite CSV retrieved from NASA's Open Data Portal, data.nasa.gov/Space-Science/Meteorite-Landings/gh4g-9sfh.