There are three problems this week, the first two dealing with SQL grouping, and the third dealing with Docker images. For the first two problems, the repository contains a zip file which can be used to recreate a collection of tables of airplane flights over the month of January, 2020. If you are interested in more, similar type data, I grabbed the original data from the Bureau of Transportation Statistics here.

As per usual, you should follow the link here to accept the assignment and get access to the repository:

Assignment link: https://classroom.github.com/a/pcT-OwVk

In the repository lives a zip file named `hw6.zip`. You should download that zip file and extract it, which has inside two SQL files (compressing made them small enough to upload to GitHub). These `.sql` files are a bit different from what you have seen in the past though, as they are technically a *database dump*. These are a common way to generate exact duplicates of the contents of a database. *However*, interacting with them can have some differences. The easiest, and fastest, method is to use the command line (terminal) tool `psql` that should have been installed along with the rest of PostgreSQL at the start of the semester. If you open a terminal (or command prompt) at the location of the `hw6_fast.sql` file (or open a terminal anywhere and then navigate to that location), the command that you want to run is simply:

```
psql -d analysis -U yourusername -f hw6_fast.sql
```

where `analysis` is presumably the name of the database where you've been placing everything, and `yourusername` is whatever user name you've been using with your database. If you happened to set up the database server on a non-default port, you'll also need to include `-p portnum`. If all goes successfully, this will create a new `hw6` schema in your `analysis` database and set up the 6 different tables, complete with data, constraints, and primary and foreign keys. You should be able to confirm this by opening whatever client you usually use and querying the tables.

If this is not working, for whatever reason, I've tried to give you another option as well. The `hw6_slow.sql` file is also a database dump, but one which has been tweaked to use standard `INSERT INTO` statements. The result is that you should be able to run this file like any other collection of SQL commands in your client, but the price is that it will be MUCH slower to populate the initial tables. I'd recommend the first option if possible, but if you are having absolutely no luck getting `psql` to work in the terminal, I think this will work. *If you are still having trouble getting the tables generated, please reach out and I'm happy to help you through this!*

The benefit of loading a database dump, of course, is that the database structure is all ready for you, with a central table `flights` having information about individual flights as well as references to other tables with information about airports, carriers, and city markets (to name a few). An image of the ERD is included on the next page, as well as in the repository, for your reference, or you can look at the version online here.

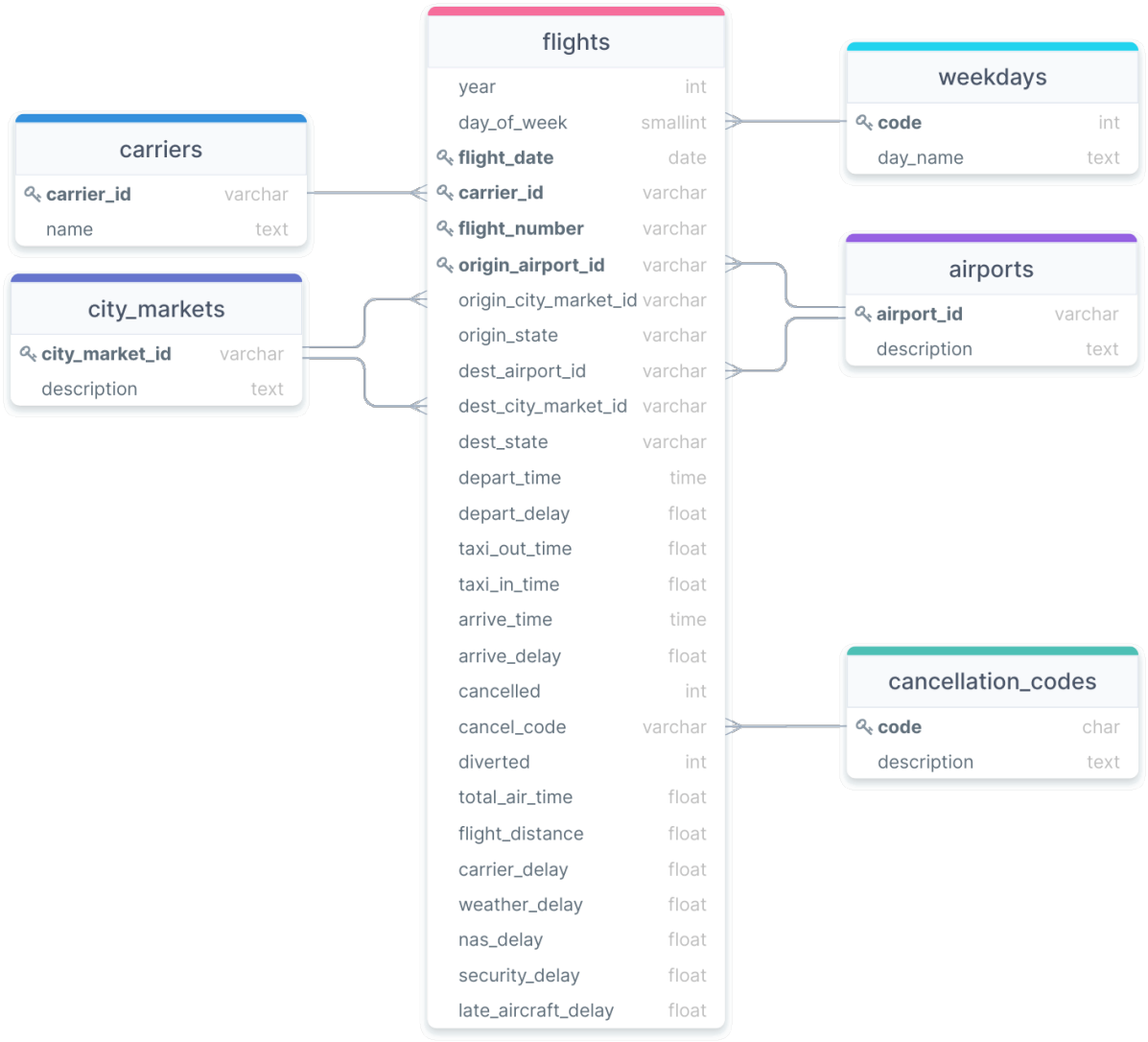

Figure 1: Entity Relationship Diagram of the tables and relationships included in the `hw6` database. The `flights` table is the main table, and references several smaller tables for extra details (mainly names) of carriers, airports, and city markets (the metropolitan area that a particular airport serves).

1. This initial batch of questions will focus on being able to use keywords such as `GROUP BY` effectively. For each you should **include any queries you used** to determine the answer, as well as the answer itself. *You may need several queries or temporary tables to answer some of the questions. Show all your work.*

   (a) What is the name of the airport which, over the given month's data, has the greatest average delay in arriving flights? Limit your answer to only include airports with at least 100 arriving flights over the course of the month.

   (b) Which 3 US cities have the greatest number of inbound flights across all airports serving that city? (Some cities have multiple airports, which is what the `city_market_id` is supposed to assist with.)

   (c) Which 5 airports have the greatest number of outgoing flights *per day* on average? How many flights (on average) are leaving those airports each day?

   (d) For each major airline carrier, which two airports represent the endpoints of that carrier's longest flight (by distance)? There are 17 distinct carriers in the month represented in the dataset, so your output table should also have 17 rows with the name of the carrier and the name of both airports. Dealing with round trip duplicates here can be a little tricky, so think about how you could reliably filter out one of the directions. It doesn't matter whether you choose PDX to SEA or SEA to PDX (for instance), you just need one of the pairings for each carrier. You can just upload a CSV of this table back to GitHub if you like, instead of including all the information in the Markdown file (though you can create tables in Markdown).

2. Just looking at the numeric outputs from our queries isn't always the most enlightening. Sometimes a little data visualization goes a long way. Depending on what you have done in the past, you may have a host of data visualization tools in your toolbox, ranging from R through Python to Excel. Any are completely appropriate for the following questions, but regardless of what software you use, ensure that your charts have clearly labeled axes and a title describing what they depict. You may need to save the output of your query in a format that your visualization software can understand (frequently CSV). Remember that you already *know* how to do this, and it is very similar to how you import information into a table.

   (a) Create a simple bar chart showing the total number of departing flights each day of the week.

   (b) This second part is more open-ended. Define a question that you are interested in investigating in this current data set. There is lots of potential data here that I haven't already asked questions about that might be interesting! The only requirement is that you must use `GROUP BY` in the course of answering your question, and you must include a visualization to help you answer the question. Clearly state both your question, your work to answer the question, and your visualization in your submission.

3. We recently introduced the concept of Docker containers, and this problem will involve both creating and then running a Docker container. As such, if you haven't already downloaded and gotten Docker running on your computer, you will need to start with that, as described in the lecture slides.

   The first part of this question will involve creating a Docker image of your own. In the `Docker_Materials` folder in the repository, create a file with the name `Dockerfile`. Inside, following the conventions shown in the slides, you will need to add the necessary lines to build your own custom Docker container. For this container, you will need the following (not necessarily in this exact order):

   - Your base image will be the latest ubuntu image, which gets you a full Linux environment at only around 150 MB.
   - It is always a good idea to set a working directory for you to copy files into and work inside. You can follow what we did in class and name in `/app`, or give it a different name of your choosing.
   - You will need to copy the obfuscated `cmd` file from the local folder into the desired folder in the image
   - This file requires some programs that are not installed by default in Ubuntu. As such, you'll need to have the image grab them when built. Ubuntu, like many flavors of Linux built on the Debian base, utilizes a package manager / installer called *aptitude*, which is interacted with through the use of `apt-get`. Before you can install anything with `apt-get`, you need to update it:

     ```
     apt-get update
     ```

     These are commands that need to be run inside the image, and so you should be prefacing them with the `RUN` keyword. Then you will need to install the two needed packages. Here the `-y` flag says to automatically accept any prompts, which is necessary when building things automatically.

     ```
     apt-get install -y figlet lolcat
     ```

   - `lolcat` gets installed in an odd location in Ubuntu, which is not in the default `PATH`. As such, we need to add its location to the path environment variable. Setting environment variables in Dockerfiles is done with the `ENV` keyword, so the whole line would look like:

     ```
     ENV PATH=/usr/games:$PATH
     ```

   - The program needs to be told what command to run when the container is launched. In this case the `cmd` script can just be launched with `bash`. Remember though that in the command portion of the docker file you give it a list of commands and options inside square brackets.

   Then you can build your image! When building your image, I'd strongly recommend adding a tag so that it has a full name + tag:

```
docker build -t yourname:latest .
```

assuming you are currently in the same folder as the Dockerfile. If all goes as planned, you should see it working through the process of building your image!

Once the image is built, launch a new container using that image! I'd strongly recommend using the `-it --rm` flags so that old containers will be cleaned up for you. You should see something colorful printed to your terminal screen! **If you do, take a screenshot and upload it back to GitHub along with your Dockerfile to submit this problem.** If you do *not*, or you get some sort of error, go back to your Dockerfile and investigate what you may need to change. And then remember once you change something you will need to rebuild the Docker image before trying to launch another container using it.

I've tried to lay this out fairly clearly and to attempt to foresee likely issues, but if something in the above process is utterly failing or making no sense, please reach out with questions. This is a newly created problem, and I can't guarantee I've foreseen every possible issue that could come up with everyone's varied architecture.