C4M Phase III, Fall 2017

Project 3: Working with a Database

Learning Goals

By the end of this project you should be able to:

- write basic SQL queries to access an existing database including using aggregation and views
- insert new values into an existing database – including realizing the implications of constraints
- use PostgreSQL document to look up details and use existing functions
- read an existing schema for a database and design research questions that could be answered by using an SQL query
- appreciate ethical considerations in working with databases of real patient data
- (optionally) complete your CITI Data or Specimens Only Research certification to be allowed access to research data

Software

To complete this assignment, you will need PostgreSQL. You can either download it and install it on your laptop or simply use it on our CS teaching machines. We recommend that you do the latter and will give instructions here for this approach.

The starter files are provided in a single zip file on the C4M site and also available at `/u/mcraig/C4M/C4MProject3.zip`. To set up your database, copy the zip file to the directory in which you want to do your work and unzip it. To do this, open a terminal window and run the following commands on the CS teaching network:

```
cp /u/mcraig/C4M/C4MProject3.zip .
unzip C4MProject3.zip
```

Next start PostgreSQL (using the command below but replacing *mcraig* with your own login):

```
psql c4mws-mcraig
```

Now you are ready to load the schema and data, which you will do in a later step.

MIMIC-III

This project is based on the MIMIC-III database which includes data from more than 58,000 hospital admissions from June 2001 through October 2012. The data includes records for 38,645 adults and 7,875 neonates. The database has 26 tables, most of which have many attributes. If you complete this entire project, including the optional steps, you can run your own queries on the actual database and try to answer a question of your choice. However, working with this anonymized yet still-sensitive data requires that you complete a short online research-ethics course and certification. Also, since the tables are large, running your queries can take a long time. For these reasons, the first steps of this assignment involve only accessing the public schema for this database and then running queries on a simplified version (C4M-MIMIC) with only a few tables, a smaller number of attributes, and many fewer records. The data in our C4M-MIMIC database are all fake.

Understanding a Schema

Start this project by exploring the MIMIC-III schema from the documents link at `https://mimic.physionet.org/about/mimic/`. Read the overview from the Getting Started menu and then explore the schema for some of the tables. Under `Data Details -> Schema` you will find a link to a graphical tool for viewing a database schema called SchemaSpy. Use SchemaSpy to further investigate the data that are available in MIMIC-III.

Using either the table descriptions or SchemaSpy, answer the following thought questions. You do not need to hand in answers for these.

- Where is the birthdate of a patient stored?
- Where is the ethnicity of a patient stored?
- Can a patient be admitted to the hospital more than once?
- What is the difference between the diagnosis in the `Admissions` table and those in the `Diagnoses_ICD` table?
- Does every `ICUStay` have an `Admission`? Does every `Admission` have an `ICUStay`?
- How does the database store information about a particular patient's prescriptions?
- How do we know the age of a patient when they were admitted? Is it stored somewhere?

Our Simplified Schema - C4M-MIMIC

Now read the files we are using to create our simplified database starting with `C4M-MIMIC.sql` which creates the tables. Start PostgreSQL (either on your own laptop or using your teach.cs account on dbserv1) and from the PostgreSQL prompt, create the mimic tables by running `\i C4M-MIMIC.sql` to run the commands in this file. Next use `\d` to list all the tables and `\d <table name>` to explore a particular table. Run `\i C4M-MIMIC.data` to load the fake data. Continue to explore some of values using some simple `SELECT * FROM <table name>` queries to confirm that you have correctly set up the database.

Try adding a new Patient and then a new Admission. Notice that you can't add an Admission for a Patient who isn't in the database and that you can't add a new Patient with a subject_id that has already been used.

Answer the following thought questions. You do not need to submit answers to these.

- Which tables are created in the simplified C4M-MIMIC schema?
- How are rows from one table connected to rows from another?
- Which fields were in the real MIMIC `Admissions` table but dropped from the simplified version?
- Do any of the remaining tables have all the same fields as their non-simplified versions?
- In the simplified version, can a patient have more visits to the hospital than trips to the ICU? Can they have more trips to the ICU than visits to the hospital?

Part 1: Writing Queries

In this part of the assignment, you will write eight queries that you will test against the fake data. To demonstrate that your queries are working, put them in files named `q1.sql` through `q8.sql`. . Sometimes the instructions below include hints about how to develop the queries in incremental steps. In this case, you need only submit your final version. However when your query involves using creating a new view (or multiple views), be sure to include any `CREATE VIEW` queries in your file. Start a fresh copy of the C4M-MIMIC database and run your final `q1.sql` throught `q8.sql` files from the PostgreSQL prompt. Cut and paste your entire interaction with the PostgreSQL shell into a plain text file named `demo.txt`.

1. Let's explore the different admission types in the data. How many different types are there? What are the different types? How many admissions are of each type?

   Write a query that for each different admission type, prints the name of the admission type and the number of admissions of this type.

2. Let's explore the different diagnoses that are associated with admissions. First let's count how many different diagnosis codes show up in the data. Write a query to simply output the number of **different** diagnosis codes were actually used in the data.

3. We would like to know which diagnoses happen most often. Write a query to show the number of admissions that have each diagnosis code. Since we are interested in the common diagnoses, order the output by decreasing count (so that the ones that happen most are first.) In fact, we aren't really interested in any diagnosis that doesn't occur at least 200 times, so only display the codes for diagnoses that happen this often.

4. The ICD9 code isn't really that informative, so instead of displaying the codes for the diagnoses, change your previous query to display the short title for each diagnosis. HINT: Think about where the information is that you need to solve this problem and where it is stored. If you have trouble because you can't display the title in the same query as a `GROUP BY`, use an intermediate view.

5. Find the names of the most common diagnoses in the dataset. You can do this simply by ordering the previous query and reading the result for the highest count or if you want a harder challenge, write a query to return only the name of the most common (or all the diagnoses tied for most common in the event of a tie.)

6. We are interested in the birthdates of patients who have had pneumonia. You might notice that in the table of names for diagnoses (`d_icd_diagnoses`), there are lots of variations of related diseases. Find all the `subject_id`'s and date of birth of patients who have had any diagnosis that includes "Pneumonia" anywhere in the diagnosis `long_title`.

7. We would like to explore something about the severity of different diagnoses. How does the simplified database represent whether or not a patient is still living? Write a query that lists the `short_titles` for diagnoses for which everyone who has ever had that diagnoses is still alive. HINT: Tackle this query in steps with views. Warm up by counting how many patients in the database are living and how many have died. Next start a new query to list diagnosis codes for which at least one patient is dead. Use that result to find diagnosis codes for which no-one is dead. Then use that list of codes to create the list of `short_titles` that you need.

8. Suppose you consider a patient more at risk of some particular disease if they are under six months old (at the time of their admission) or they are a male over 70 years old or a female over 80. Find the list of subjects who meet this criteria and list their id, age at admission (as "age") and gender. Hint: You can subtract `TIMESTAMP` attributes to get intervals and you can compare intervals as shown in these examples:

   ```
   WHERE age > '4 days'
   WHERE interval <= '2 years'
   ```

Getting Creative

Your next challenge is to find something obviously wrong with the fake data. It wasn't generated by collecting real observations but through a Python program using a random generator and some minimal thought about statistics. (You could probably have written that program with just a teeny bit of advice!) The generating program wasn't terribly sophisticated and there are some problems with the fake data. Can you find one? Explain a problem you find and the query you wrote to discover it in a file called `fake_data_problem.txt`

Part 2: Asking a Real Research Question

Your Question

Your final task is to propose a real question that you would like to answer using the real MIMIC Data. Start by looking again at the schema for full set of patient data. I recommend you don't go too deeply into the various

events files. These are probably the most interesting, but they are complicated and working with them will probably take more time than you want to invest right now. But you could easily look further at the `prescriptions`, `procedures`, or `caregivers`. You could also look at the attributes that we stripped out of the `Patients` table to make our simplified version.

Propose a question for which you would like to know the answer and then design a query to run on the real database. Once you have a question and a first attempt at the query, email Michelle (mcraig@cs.toronto.edu) to tell her your question. She will give you feedback on your query and depending on your situation, attempt to either help you work out any extra features of SQL that you might need or find a way to simplify your question to something that you **can** answer with SQL.

Debugging Your Query

Once you have a reasonable query to write, you'll need to develop and test the query itself. But if it involves tables that aren't in the fake data, you'll need to create them. Using the tables already in the file `C4M-MIMIC.sql` as a model, add the `CREATE TABLE` statements that you need. Your new table isn't going to have any data so use `INSERT` statements to add just a few rows of test data. If your query is going to need additional attributes from one of the simplified tables used in the initial fake data, you'll want to add those new attributes into `C4M-MIMIC.sql` and into the corresponding CSV (comma separated value) file where we put the initial fake data. Ask if you need help with this!

Running Your Query

Now that you have a query that runs correctly on fake data, you have two choices for running it on the real MIMIC-III data. Ideally, you should complete the CITI Data or Specimens Only Research course and apply for your own PhysioNet account. If you do that, list Michelle as your reference on the application. Notice that it takes about a week from when you apply (after getting your CITI certification) before you'll hear back and have access, so don't expect to do this step and then work with the real data on the same day.

Once you are authorized you can use a helpful tool called "QueryBuilder" that lets you simply type in the queries online and it runs them against the full MIMIC dataset and shows you up to the first 1000 lines of the result. If you plan to go this route, you can use QueryBuilder to do the previous debugging step and avoid having to alter the fake data schema. You can also run your first eight queries against the real MIMIC patient data and see the **actual** answers to our exploration questions!

The problem is that the time to complete the CITI 'Data or Specimens Only Research' course can be one or two hours (depending on how quickly you read) and while some of it is very interesting ethical questions, other parts are more about ethical regulation in the USA. So this part of the assignment is not required. If you instead debug your query on the extended fake data that you created in the previous step, Michelle can run your query on MIMIC and let you know the final answer (but not any of the confidential data itself.)

What to hand in

For Part 1, hand in your plain text files `q1.sql` through `q8.sql`, `demo.txt` and `fake_data_problem.txt` through MarkUs.

For Part 2, write a brief report that outlines your research question, the query you wrote to find the answer, the results from running your query on fake testing data and the results from running your query on the real MIMIC data. This is likely less than 1 page unless your query itself is very complicated. If the results are long, don't include them all – just an explanation of how they answer your research question. Save your report in either plain text or PDF and submit it through MarkUs.