STAT 19000 Project 10

Topics: Working with multiple tables

Motivation: Databases would be fairly limited if we were unable to do more with them than shown in project 9. In this project, we will learn new topics that let us fully take advantage of what a database has to offer.

Context: In the last project, we learned how to navigate a database using a database management system called `sqlite3`, and dove head first into various ways to extract information quickly and effectively. Now we will continue to learn about some more advanced SQL topics that are a critical addition to any scientist or engineer's toolset.

Scope: SQL topics including but not limited to: joins, grouping, pattern matching, having, intersect, union, & except.

You can find useful examples that walk you through relevant material here or on scholar:

`/class/datamine/data/spring2020/stat19000project10examples.pdf`.

It is highly recommended to read through these to help solve problems.

Use the template found here or on scholar:

`/class/datamine/data/spring2020/stat19000project10template.Rmd`

to submit your solutions.

Important note: From this point on, unless specified otherwise, submit your solutions as an RMarkdown (`.Rmd`) file. Place code in the provided code chunks. Submit both the raw `.Rmd` file as well as the knitted PDF.

After each problem, we've provided you with a list of keywords. These keywords could be package names, functions, or important terms that will point you in the right direction when trying to solve the problem, and give you accurate terminology that you can further look into online. You are not required to utilize all the given keywords. You will receive full points as long as your code gives the correct result.

Don't forget the very useful documentation shortcut `?`. To use, simply type `?` in the console, followed by the name of the function you are interested in.

You can also look for package documentation by using `help(package=PACKAGENAME)`.

Sometimes it can be helpful to see the source code of a defined function. To do so, type the function's name without the `()`.

Question 1:

1a. *(1 pt)* Use the `sqlite3` command to connect to the database located on scholar:

`/class/datamine/data/spring2020/5000.db`.

Use what you've learned to explore the database. Can you identify the source of the data on scholar (`/class/datamine/data/`)? "Confirm" that the suspected source of the data is, in fact, the source of the data. To do this, assume that if the number of columns and rows in the tables in the database match the number of rows and columns in the original sources of data, that the original source is indeed where we got the data.

Hint: *This is a good resource for getting the number of columns in an SQL table.*

Item(s) to submit:
- A sentence saying whether or not you were able to confirm the original data source, or not.
- A list of all bash commands used to measure the number of rows and columns of each original source of data.
- A list of SQL queries used to measure the number of rows and columns of each table in the database.

1b. *(1 pt)* If you were to use a JOIN command in SQL on this database to combine tables, which columns in each table would you use to JOIN each table with every other table? If you wouldn't directly join two tables, say so.

Item(s) to submit:
- A list of sentences that tell which column or columns would be used to join every pair of tables together.
- Example: "To join table A with table B we would use the column `process_id` from table A and the column `pid` from table B."

1c. *(1 pt)* Use the `sqlite3` command to connect to the database located on scholar:

`/class/datamine/data/spring2020/5000.db`.

For each transaction in the `transactions` table, we want to get the associated household information (using the "INNER JOIN" command) from the `households` table. These tables can be joined together using the column `HSHD_NUM` from both the `transactions` table and the `households` table. Perform this SQL query and limit output (using the "LIMIT" command) to 10 records.

Item(s) to submit:
- The SQL statement (that uses an INNER JOIN) used to solve this problem.
- The result of the query, limited to a maximum of 10 lines (using the `LIMIT` clause).

Question 2:

2a. *(1 pt)* Use the `sqlite3` command to connect to the database located on scholar:

`/class/datamine/data/spring2020/imdb.db`.

In question (3d) from project 9, we ask:

In a single query/statement, get the title information from the `titles` table and the episode information from the `episodes` table for the title with `title_id` "tt0108778". The result should be in ascending order first by `season_number` and then by `episode_number`.

Answer the same question, but this time use a join instead.

Hint: *The result should contain 236 rows.*

Item(s) to submit:
- The SQL statement (that uses a join) used to solve this problem.
- The result of the query, limited to a maximum of 10 lines (using the `LIMIT` clause).

2b. *(2 pts)* Use the `sqlite3` command to connect to the database located on scholar:

`/class/datamine/data/spring2020/imdb.db`.

Find the episode id's and ratings for Game of Thrones (tt0944947).

Hint: *The result should contain 73 rows.*

Item(s) to submit:
- The SQL statement (that uses a join) used to solve this problem.
- The result of the query, limited to a maximum of 10 lines (using the `LIMIT` clause).

2c. *(2 pts)* Use the `sqlite3` command to connect to the database located on scholar:

`/class/datamine/data/spring2020/imdb.db`.

Get the titles of all tv episodes where the rating is less than or equal to 6, and the `show_title_id` is "tt0944947". Use join(s) to accomplish this. These are episodes that are not very good. If there is not a rating match for the episode, exclude the entire record.

Note: *Titles in the `titles` table can be referring to either an individual episode of a tv show, or the name of the tv show itself. This is why the `episodes` table has two columns: `episode_title_id` and `show_title_id`. The former will get an episode's title from the `title` table, and the latter will get the title of the tv show itself. The `ratings` table is the same as the `titles` table in that it contains ratings for individual episodes as well as the show itself.*

Hint: *You can start with a subquery with the `episodes` and `ratings` tables, then add the titles using the `titles` table.*

Hint: *The result should contain 3 rows.*

Item(s) to submit:
- The SQL statement (that uses a join) used to solve this problem.
- The result of the query, limited to a maximum of 10 lines (using the `LIMIT` clause).

Question 3:

(2 pts)

Use the `sqlite3` command to connect to the database located on scholar:

`/class/datamine/data/spring2020/chinook.db`.

In a single query/statement, get all of the track data (from the table `tracks`), and any associated invoice items for each track (from the table `invoice_items`). If there are no matching invoice items, no worries, but we still want to get track info.

Hint: *The result should contain 3759 rows.*

Item(s) to submit:
- The SQL statement (that uses a join) used to solve this problem.
- The result of the query, limited to a maximum of 10 lines (using the `LIMIT` clause).

Project Submission:

Submit your solutions for the project at this URL: https://classroom.github.com/a/EjCjO-vu using the instructions found in the GitHub Classroom instructions folder on Blackboard.

Important note: Make sure you submit your solutions in both .Rmd and .pdf formats. The PDF should be the knitted result of the .Rmd. Make sure and double check that the PDF looks okay and displays your solutions correctly.