Joshua Catoe

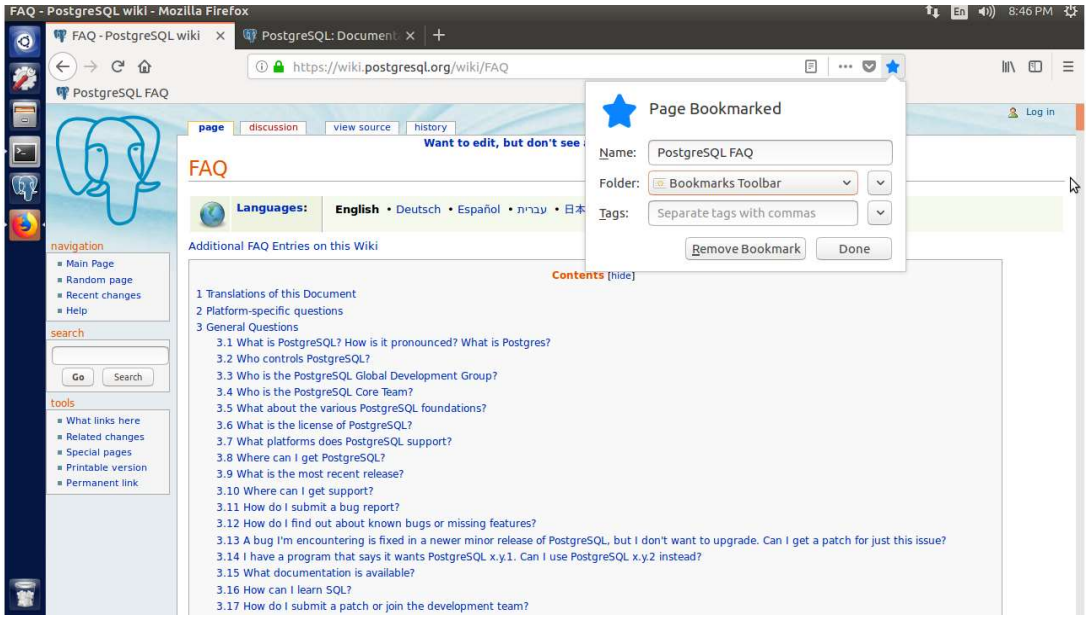
07/22/18

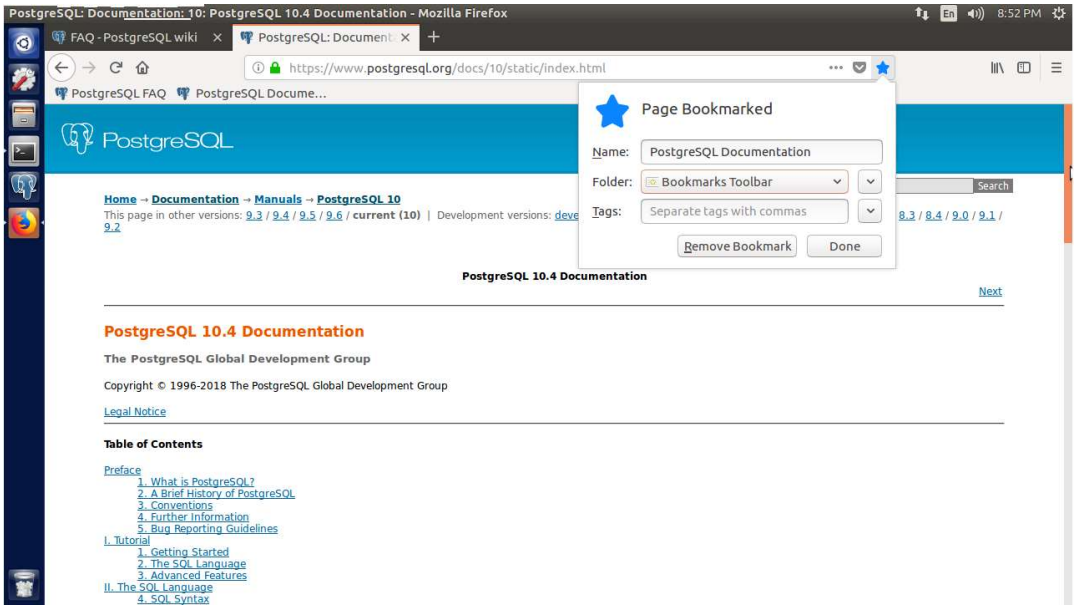
PostgreSQL

Day 1

Find:

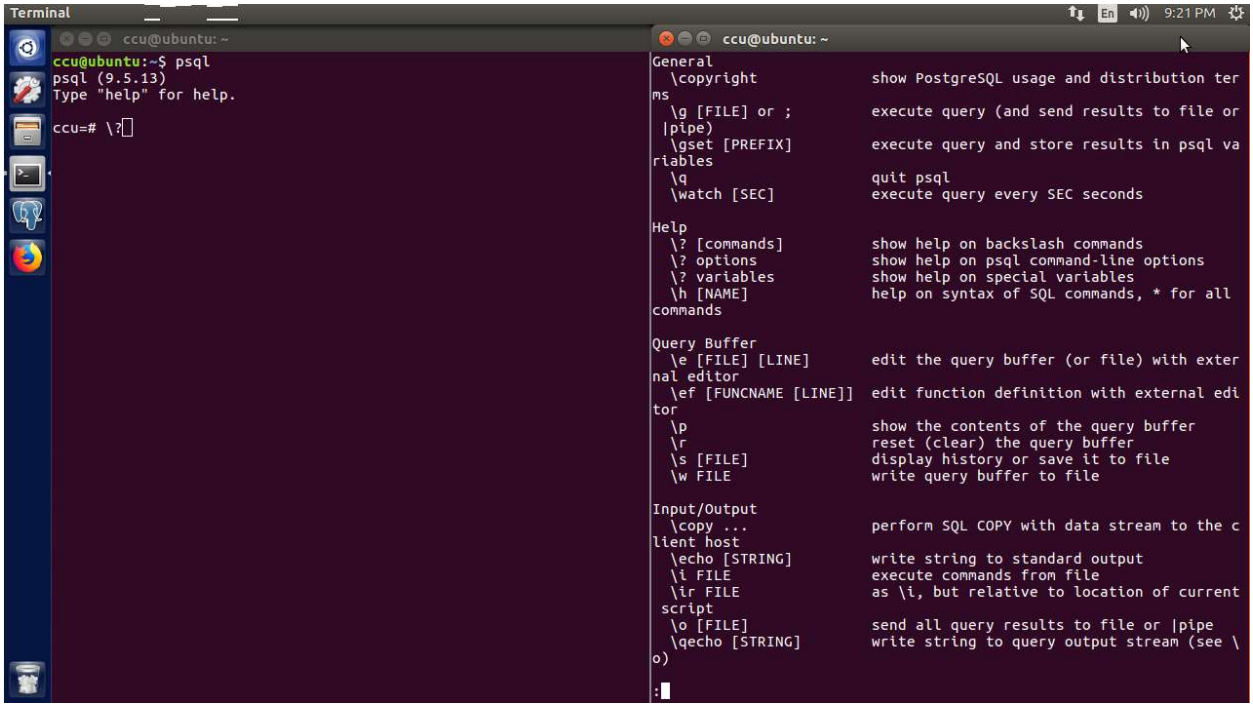
1. Bookmark the online PostgreSQL FAQ and documents.

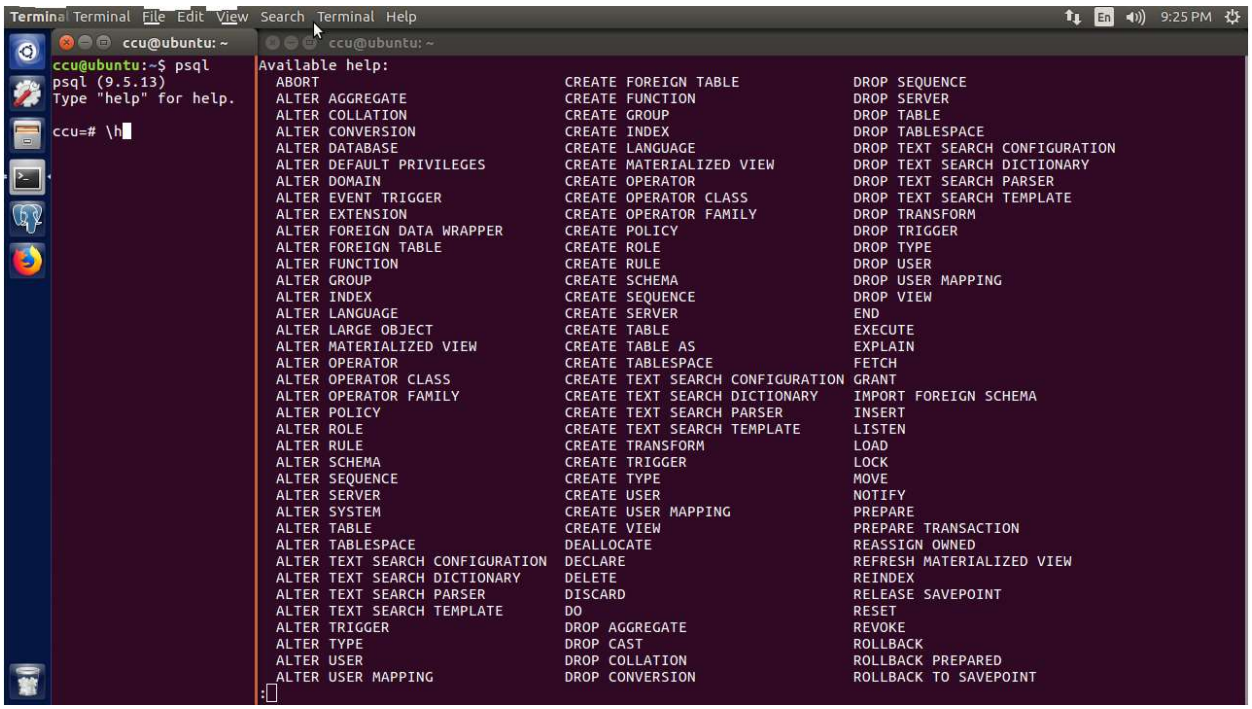




For this I simply pressed *Ctrl+D* in Firefox and added the bookmarks to a folder of my choice.

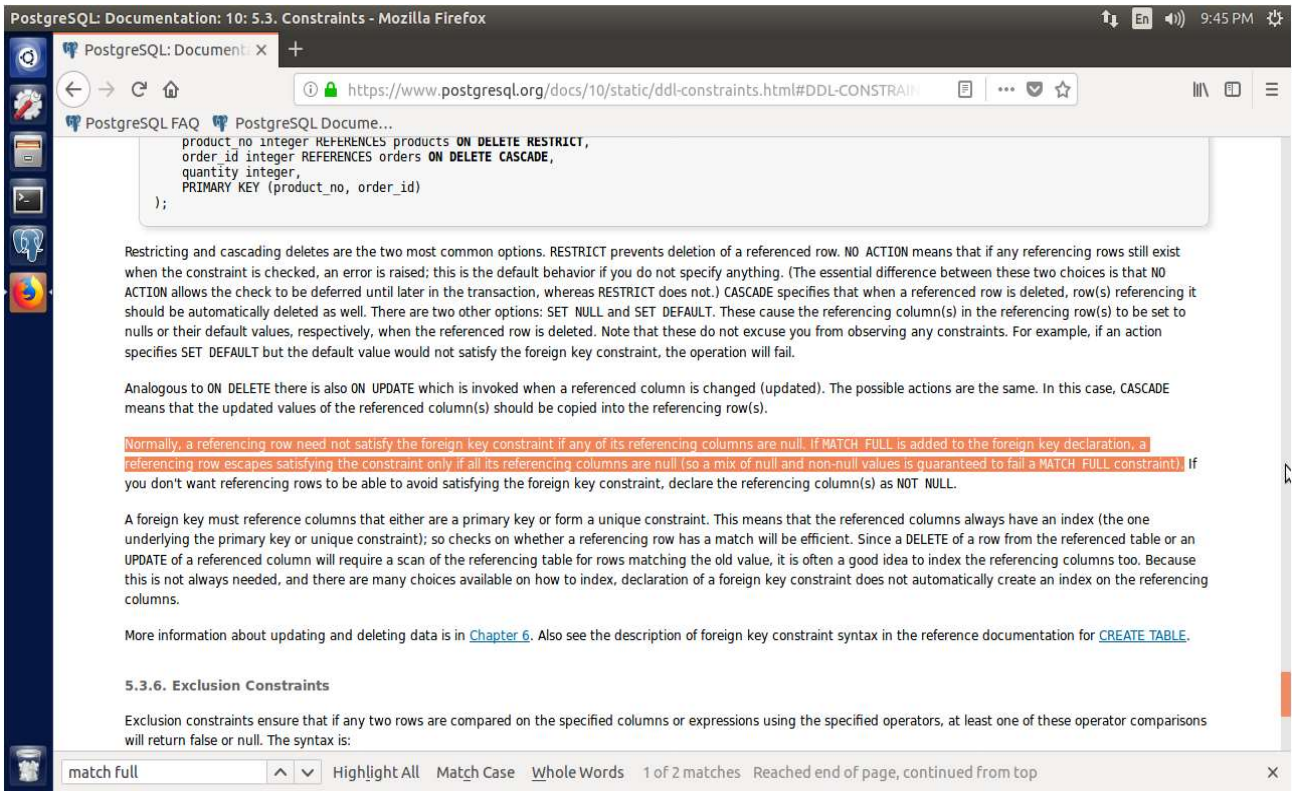
2. Acquaint yourself with the command-line **\?** and **\h** output.





In the terminal, I typed **psql** which opens a PostgreSQL shell for a database. I then entered **\?** for psql command help and **\h** for SQL command help.

3. In the addresses **FOREIGN KEY**, find in the docs what **MATCH FULL** means.

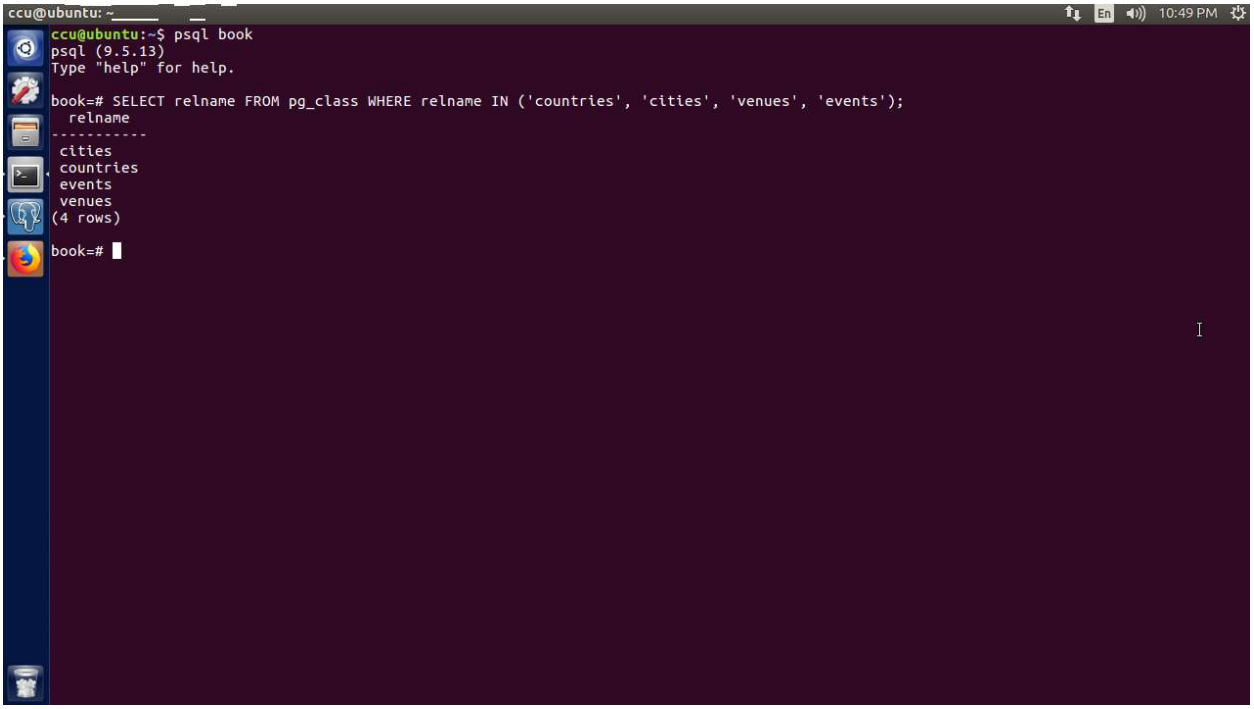


To find the use of **MATCH FULL**, I went to the PostgreSQL [10.4] Documentation bookmark and followed this series of links: II. The SQL Language -> 5.3 Constraints -> 5.3.5 Foreign Keys and then used Ctrl+F to search the page for **MATCH FULL**.

According to the documentation, **MATCH FULL** means that a referencing row can avoid referential integrity if its referencing columns are all **null**.

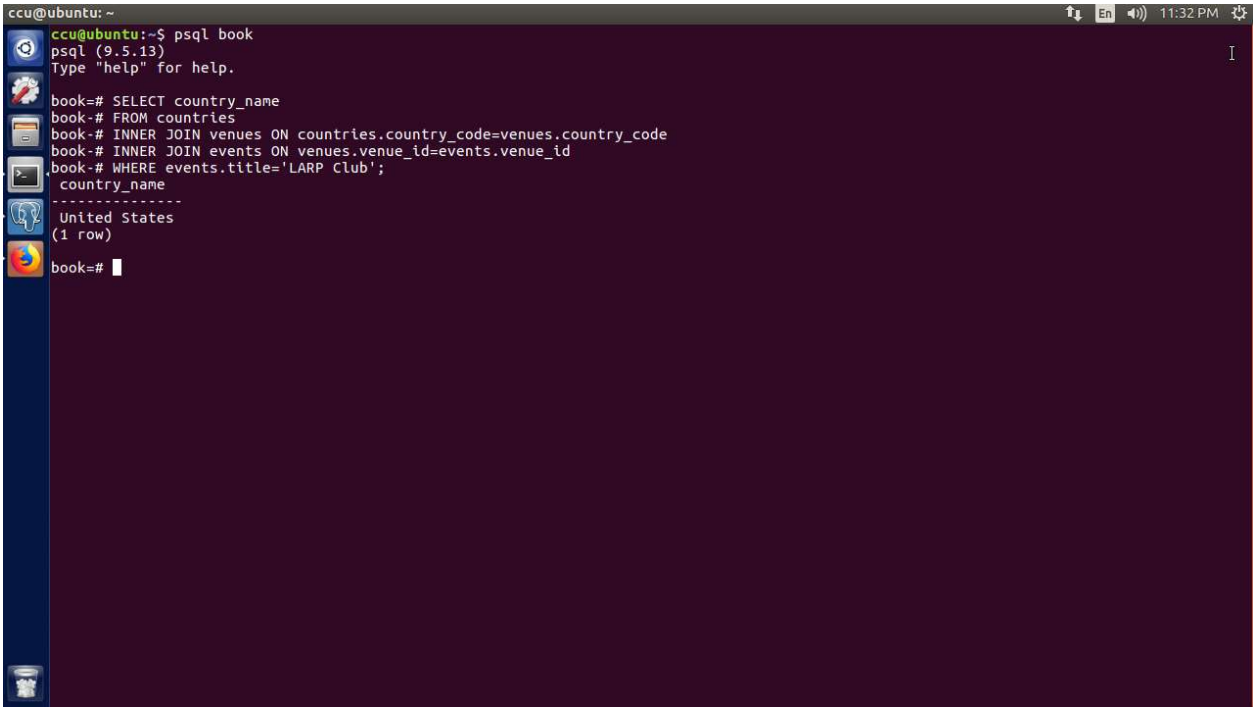
Do:

1. Select all the tables we created (and only those) from **pg_class**.



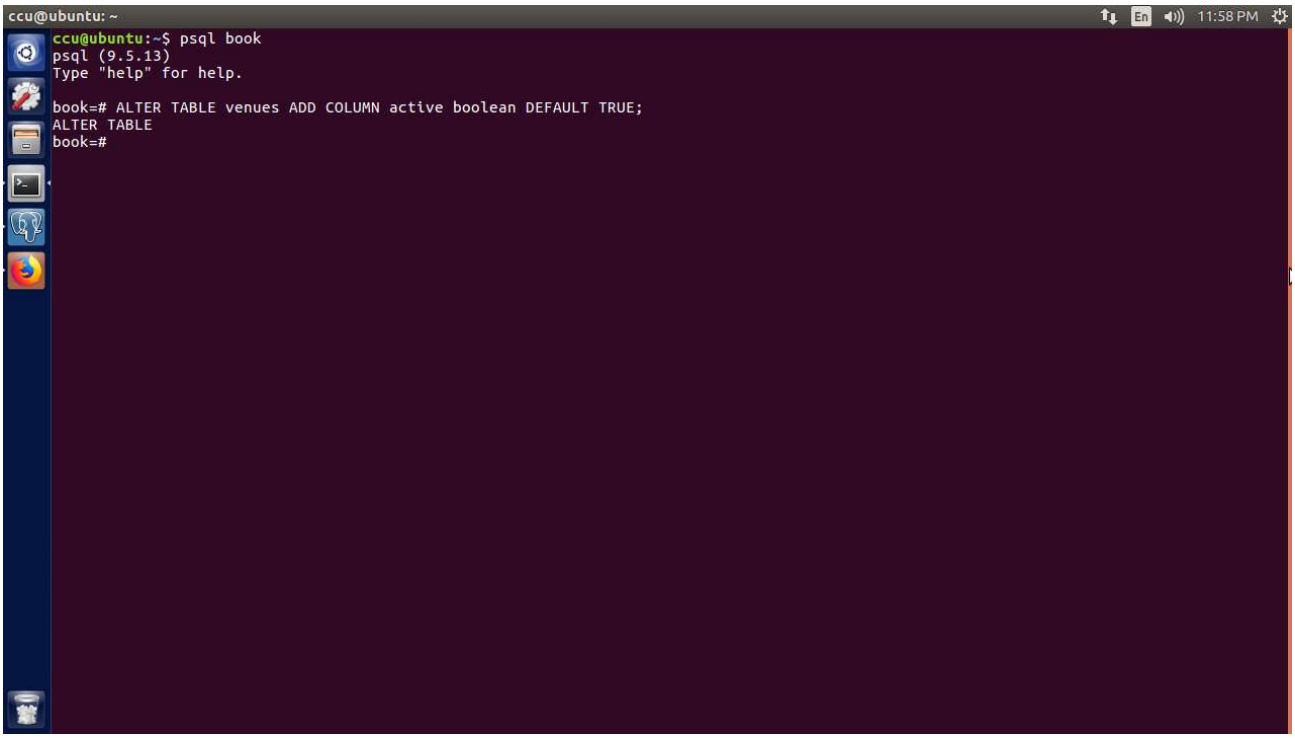
First, I had to read the documentation to find out what **pg_class** is. Apparently, **pg_class** is a catalog that catalogs tables and anything that resembles a table. One of the columns in the catalog is **relname**, which contains the names of all the tables within **pg_class**. So I took this variable and used it in a basic query to list all of the tables in the **book** database.

2. Write a query that finds the country name of the LARP Club event.



This involved a query with two simple inner joins. Column **country_code** connects tables **countries** and **venues**, and column **venue_id** connects tables **venues** and **events**. All that was needed were two inner joins to connect the three tables.

3. Alter the **venues** table to contain a Boolean column called **active**, with the default value of **TRUE**.

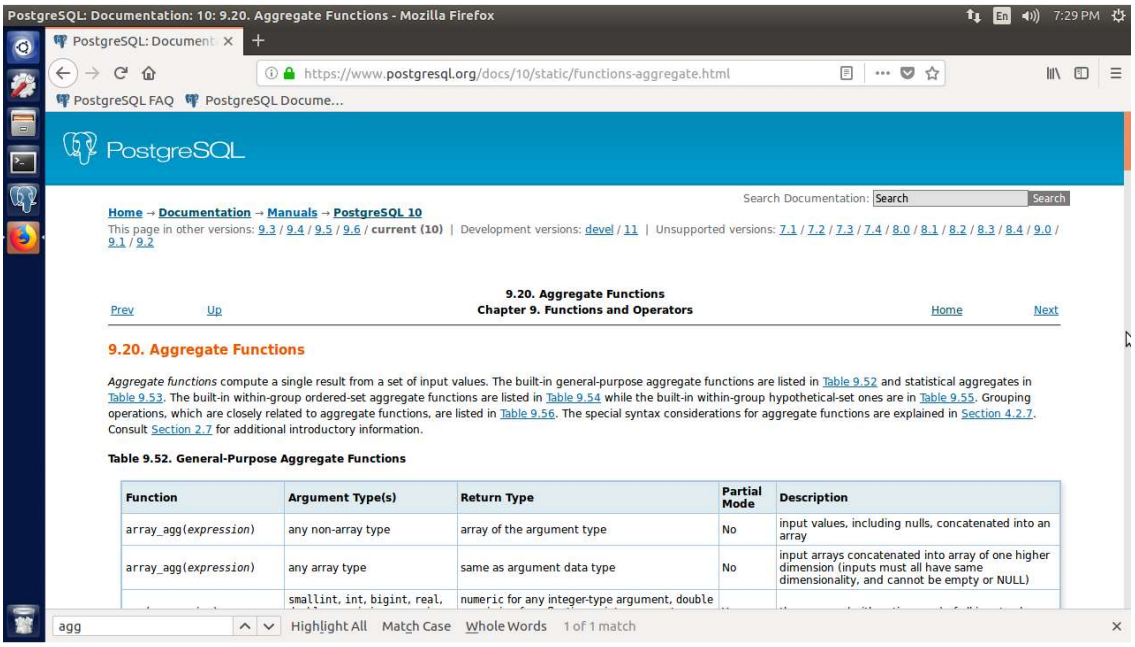


Another simple query, this one using **ALTER TABLE** to alter the table **venues, ADD COLUMN** to add the column **active** as a boolean data type to **venues**, and **DEFAULT TRUE** to set the default value of **active** to **TRUE**.

Day 2

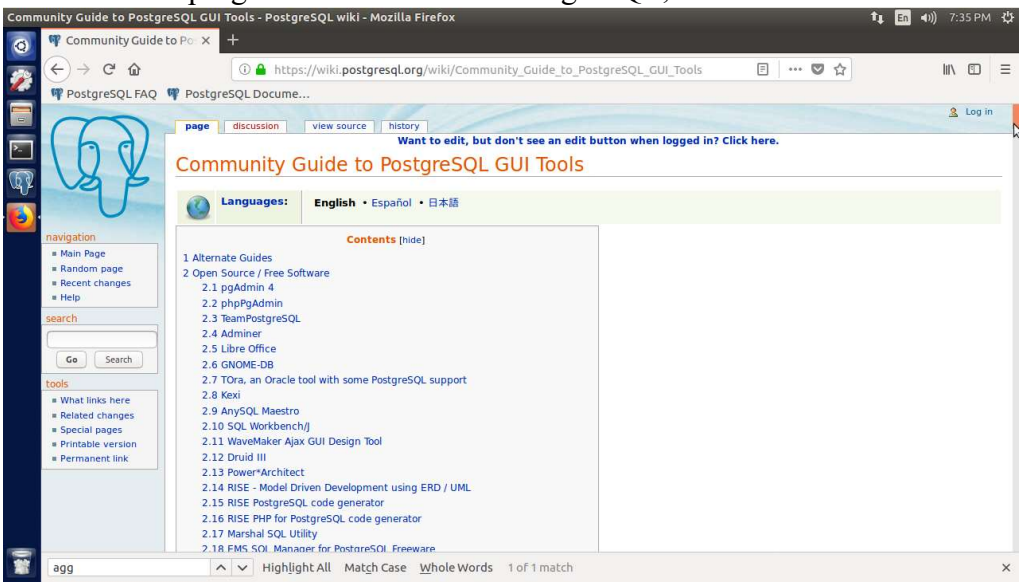
Find:

1. Find the list of aggregate functions in the PostgreSQL docs.



I simply went to the PostgreSQL [10.4] Documentation page and followed these links: II. The SQL Language -> 9.20 Aggregate Functions.

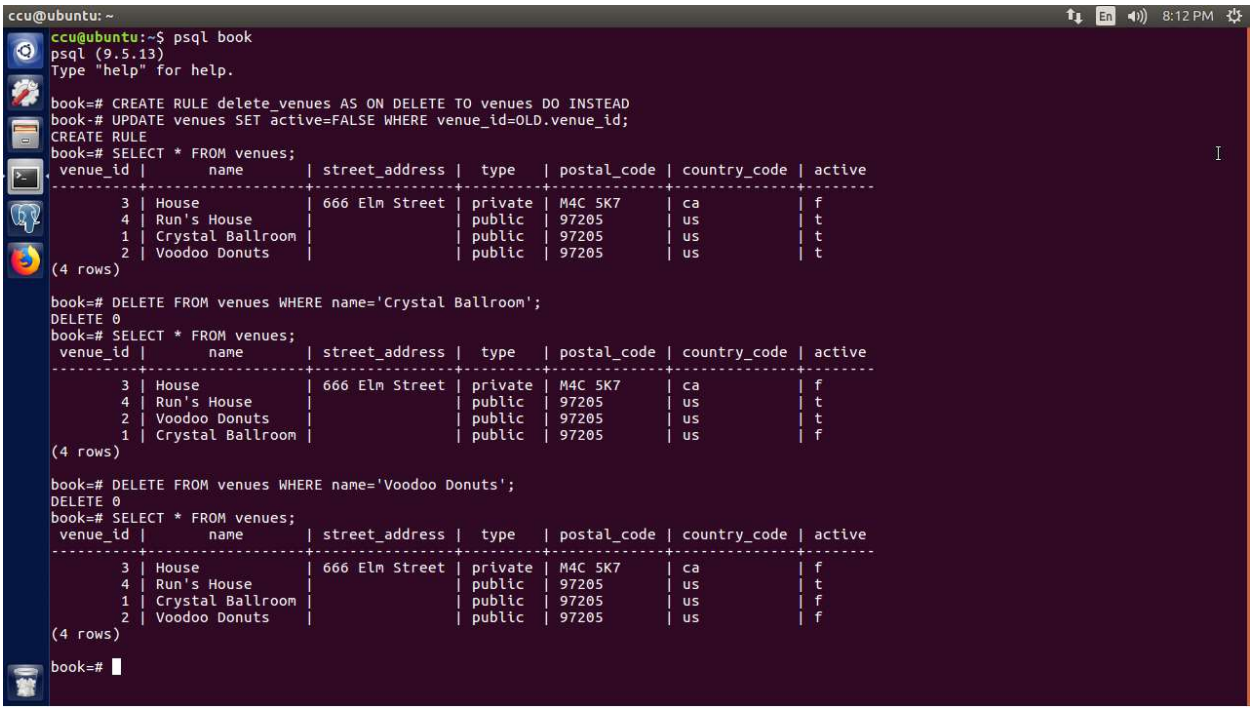
2. Find a GUI program to interact with PostgreSQL, such as Navicat.



After a quick Google search for PostgreSQL GUI tools, it turns out that the PostgreSQL wiki has an extensive listing of both open source and proprietary GUI programs for PostgreSQL.

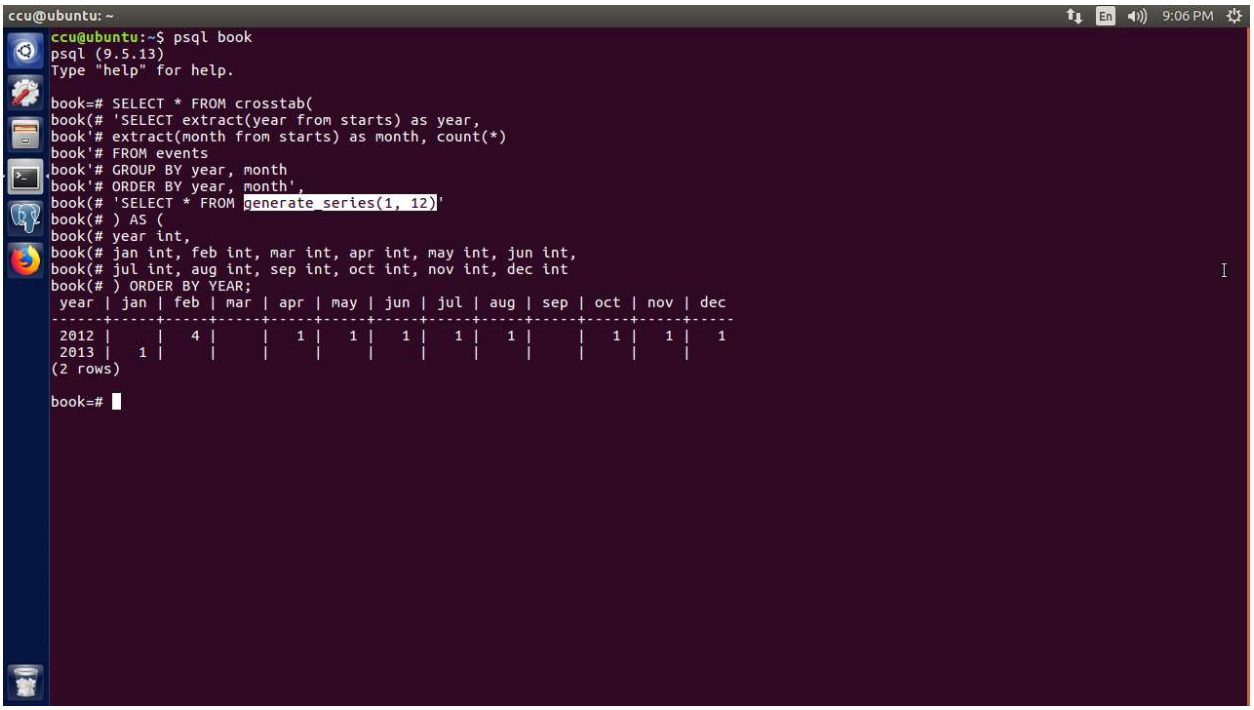
Do:

1. Create a rule that captures **DELETE**s on venues and instead sets the active flag (created in the Day 1 homework) to **FALSE**.



For this rule, I first tried to update the **active** column to **FALSE** with no other parameters, but this ended up converting the entire column to **FALSE** (which I suspected). Then, I set **active** to **FALSE** where **venue_id=venue_id**, but this resulted in the same behavior. Finally, I used **venue_id=OLD.venue_id**, which selects the **venue_id** of the row you are referring to in the **DELETE** statement. This caused the rule to work perfectly.

2. A temporary table was not the best way to implement our event calendar pivot table. The **generate_series(a, b)** function returns a set of records, from a to b. Replace the **month_count** table **SELECT** with this.

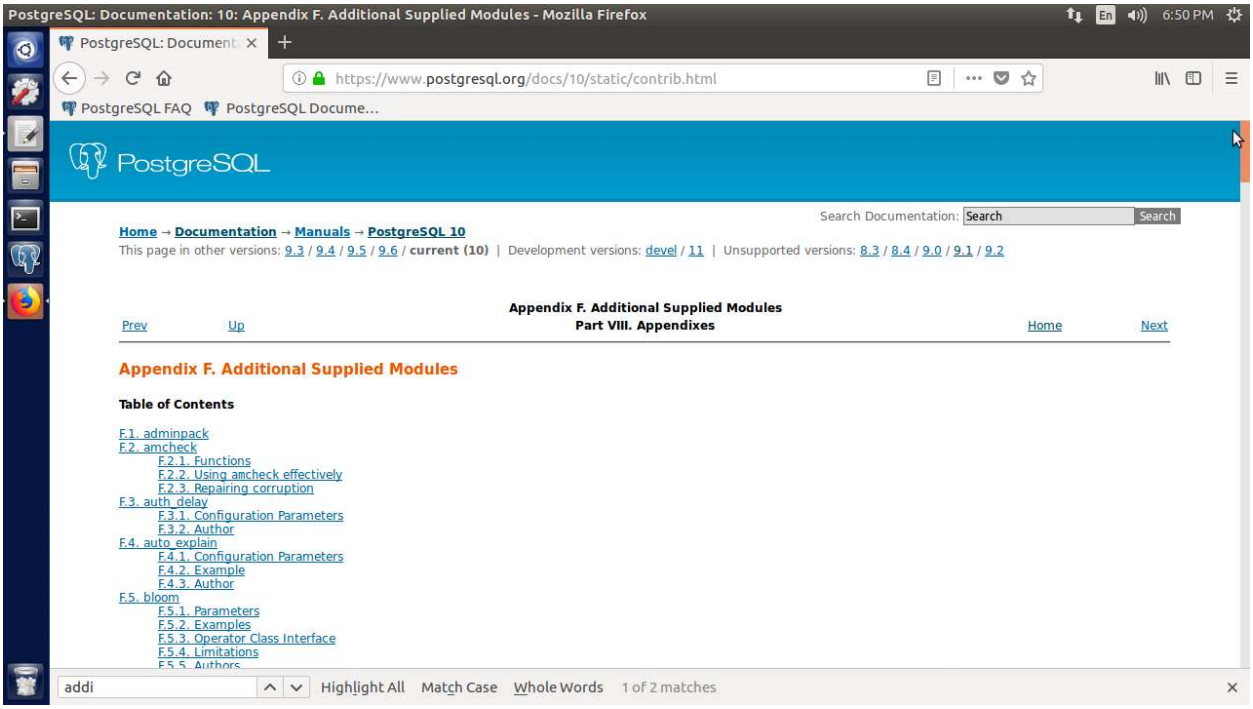


This required going to the documentation and reading about the **generate_series** function. The function **generate_series(a, b)** returns a series of integers or big integers where **a** is the start of the series, and **b** is the end. Then, I just had to replace the **'SELECT * FROM month_count'** statement in the book's example query with **'SELECT * FROM generate_series(1, 12)'**, as highlighted above.

Day 3

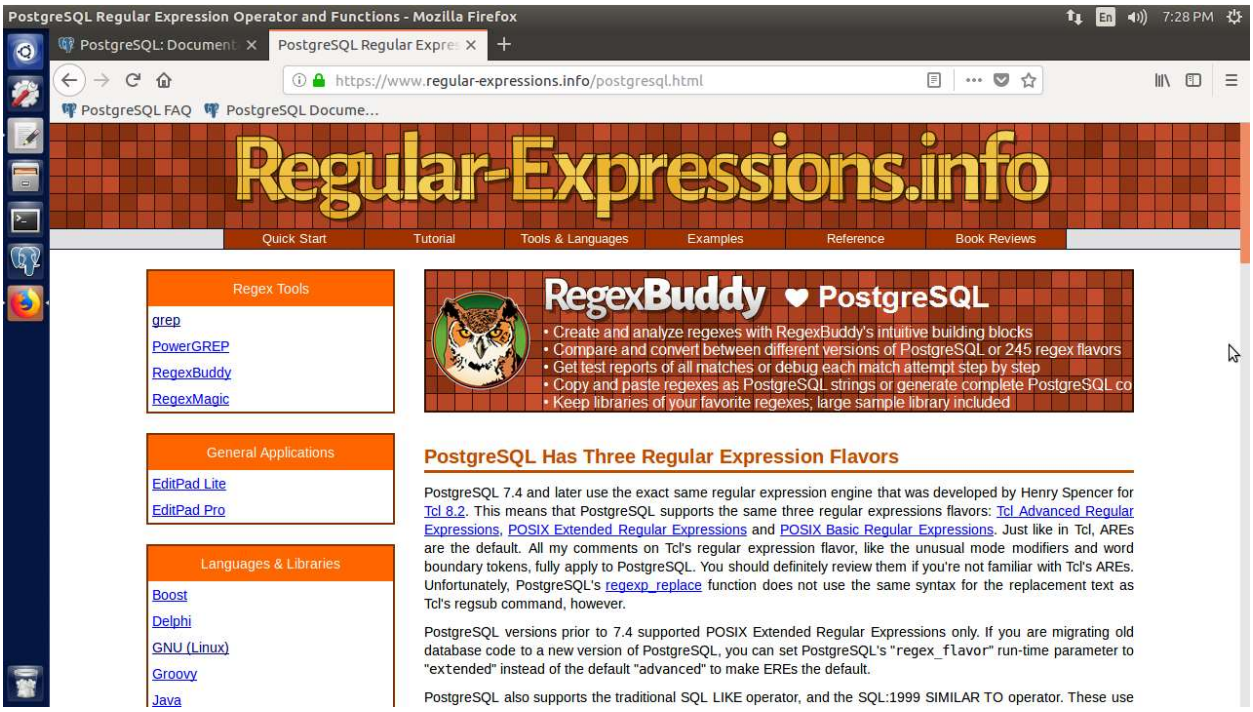
Find:

1. Find online documentation of all contributed packages bundled into Postgres.



For this, I went to the PostgreSQL documentation and followed these links: VIII. Appendixes -> F. Additional Supplied Modules. “Additional Supplied Modules” is just another name for contributed packages.

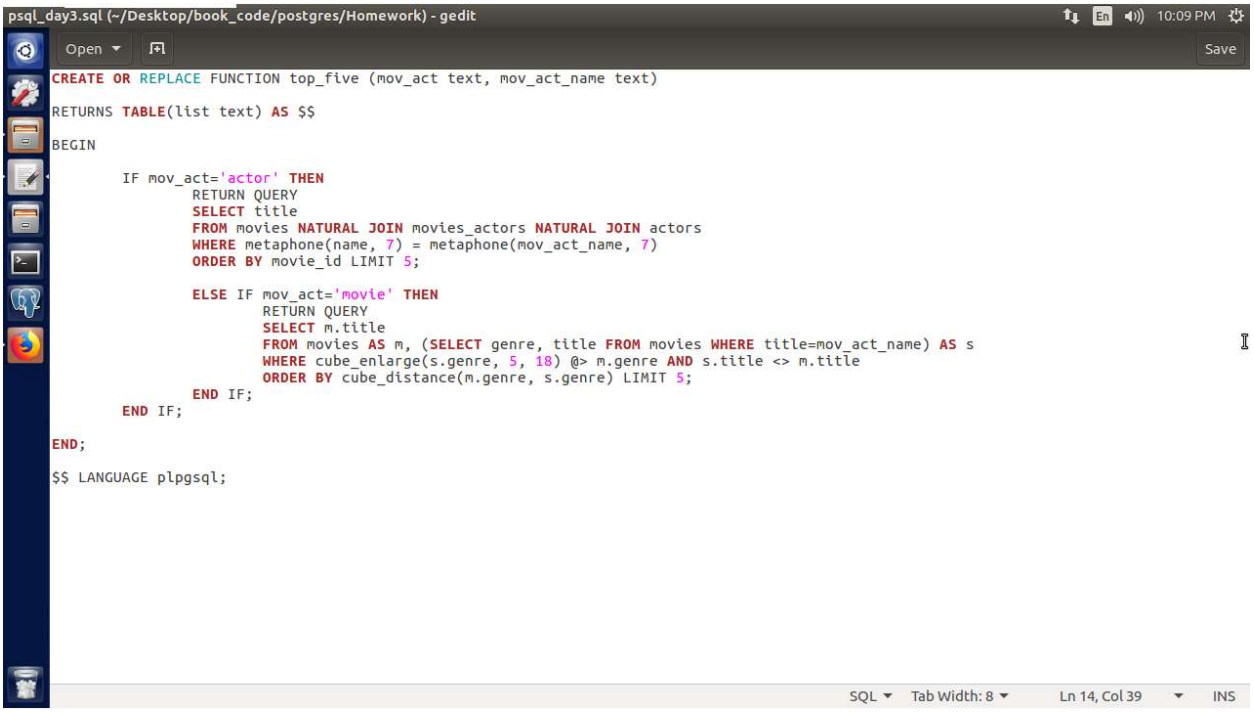
2. Find online POSIX regex documentation (it will also be handy for future chapters).

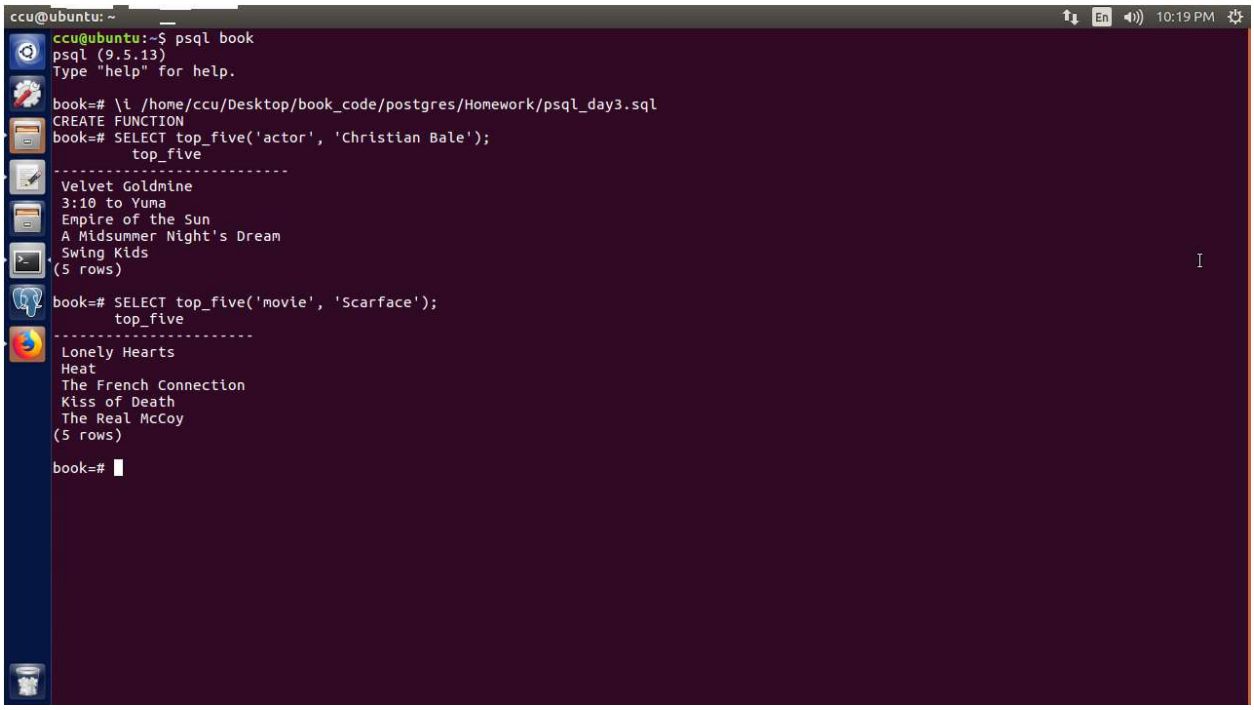


Through a Google search, I found a website called regular-expressions.info. It contains everything from quick references to in-depth guides, and also includes specific documentation for many different languages and databases.

Do:

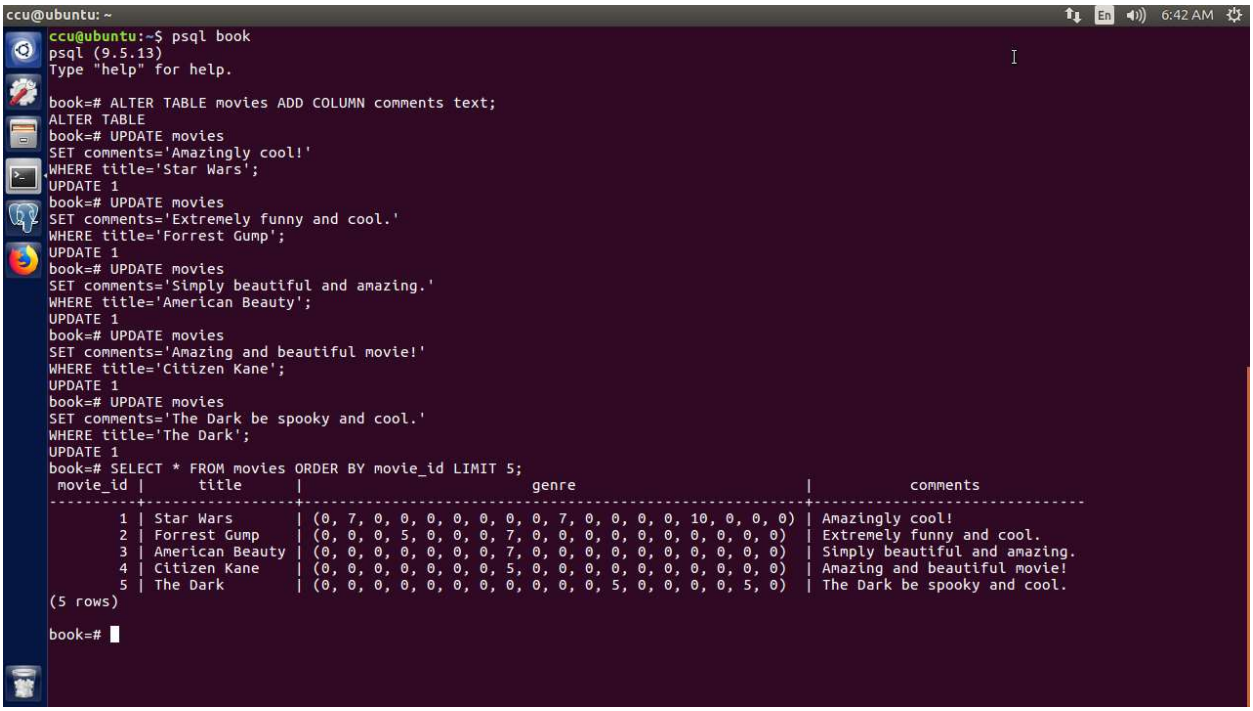
1. Create a stored procedure where you can input a movie title or actor's name you like, and it will return the top five suggestions based on either movies the actor has starred in or films with similar genres.

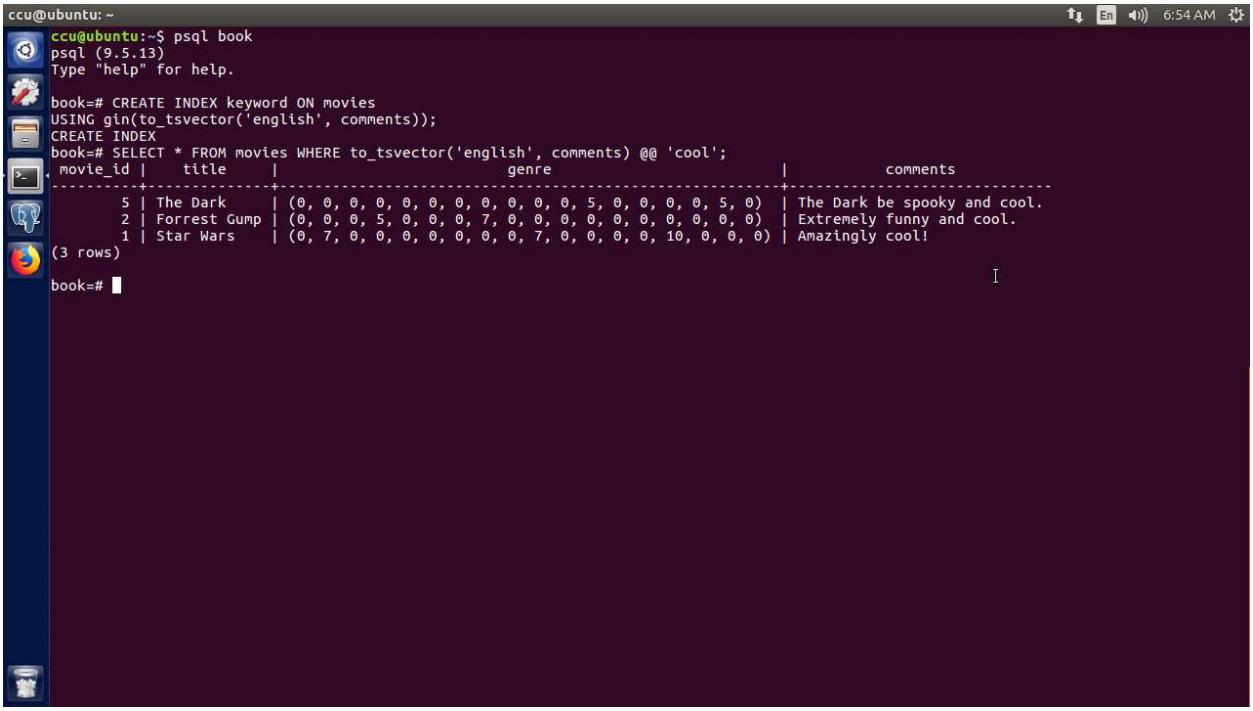


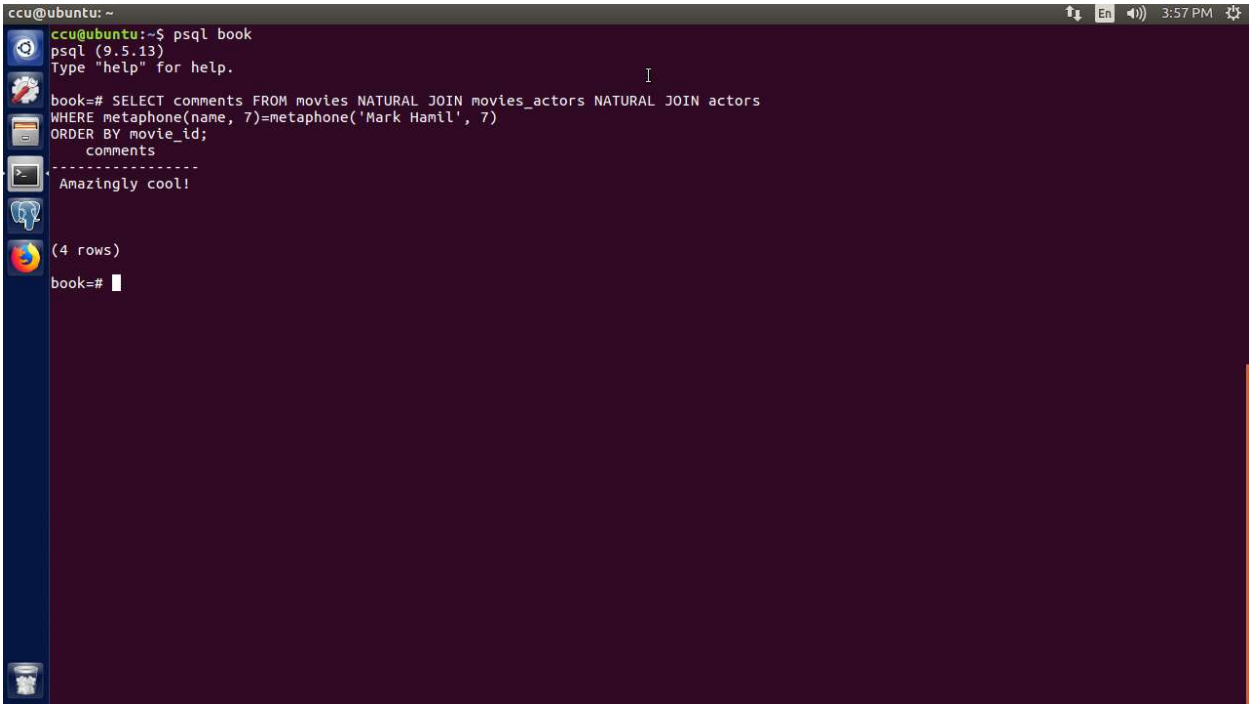


For this stored procedure, I wrote the function in a **.sql** text file and imported it into the **book** database, as this was easier than typing it directly into the command line. The main body of the function was easy, as the queries to select movies were virtually the same as the examples in the book except for one variable that needed to match the second parameter in the function. I had to research the function return types in the documentation and found out that I needed to use the **RETURN TABLE** type along with the **RETURN QUERY** statement. The **RETURN QUERY** statement returns the results of the query it refers to and **RETURN TABLE** houses the results of the query return. Thus, **RETURN TABLE** displays the list of movies that the user is looking for.

2. Expand the movies database to track user comments and extract keywords (minus English stopwords). Cross-reference these keywords with actors' last names, and try to find the most talked about actors.







To keep track of comments, I just used the **ALTER TABLE** and **ADD COLUMN** commands to add the column **comments** to the table. To extract keywords, an index was needed on the **comments** column in order to search by keyword with **to_tsvector**. Finally, cross-referencing actors with movie comments was the same as with movie titles where the tables were joined and metaphones compared.