# Database Assignment: Simple In-Database Text Analytics: Instructions

Help

If you are using the virtual machine, you will open a terminal window and type commands to complete this assignment. Some relevant commands and notes for this assignment:

- `cd directory_name` : change directory to directory_name
- `ls` : list files in the current directory
- `sqlite3 reuters.db < sql_filename` : execute the queries in the file sql_filename against the database reuters.db . This command assumes reuters.db and sql_filename are in your current directory. You can edit sql_filename with any editor, save it, and then run the query using this command.
- For Problem 2, note that you will use matrix.db instead of reuters.db

Before you start the assignment, go to the git repository and get the most recent course materials by issuing the following command: **git pull**

For most of these problems, you will use the reuters.db database consisting of a single table:

```
frequency(docid, term, count)
```

where `docid` is a document identifier corresponding to a particular file of text, `term` is an English word, and `count` is the number of the occurrences of the term within the document indicated by `docid`.

Many questions ask you to count the number of records returned by a query. Perhaps the easiest way to count the number of records returned by a query Q is to write Q as a subquery:

```
SELECT count(*) FROM (
  SELECT ...
) x;
```

Your query will be entered between the parentheses. (In SQLite, the alias "x" is not required, but in other dialects of SQL it is. So we've included it here.)

## Problem 1: Inspecting the Reuters Dataset; Basic Relational Algebra

**(a) select:** Write a query that is equivalent to the following relational algebra expression.

$\sigma_{10398\_txt\_earn}$(frequency)

What to turn in: Run your query against your local database and determine the number of records returned. In your browser, enter the number of records returned.

**(b) select project:** Write a SQL statement that is equivalent to the following relational algebra expression.

$\pi_{\text{term}}(\sigma_{\text{docid=10398\_txt\_earn and count=1}}(\text{frequency}))$

What to turn in: Run your query against your local database and determine the number of records returned as described above. On the assignment page, enter the number of records returned.

**(c) union:** Write a SQL statement that is equivalent to the following relational algebra expression. (Hint: you can use the UNION keyword in SQL)

$\pi_{\text{term}}(\sigma_{\text{docid=10398\_txt\_earn and count=1}}(\text{frequency}))\ \text{U}\ \pi_{\text{term}}(\sigma_{\text{docid=925\_txt\_trade and count=1}}(\text{frequency}))$

What to turn in: Run your query against your local database and determine the number of records returned as described above. On the assignment page, enter the number of records returned.

**(d) count:** Write a SQL statement to count the number of documents containing the word "parliament"

What to turn in: Run your query against your local database and determine the count returned as described above. On the assignment page, enter the number of records returned.

**(e) big documents** Write a SQL statement to find all documents that have more than 300 total terms, including duplicate terms. (Hint: You can use the HAVING clause, or you can use a nested query. Another hint: Remember that the count column contains the term frequencies, and you want to consider duplicates.) (docid, term_count)

What to turn in: Run your query against your local database and determine the number of records returned as described above. On the assignment page, enter the number of records returned.

**(f) two words:** Write a SQL statement to count the number of unique documents that contain both the word 'transactions' and the word 'world'.

What to turn in: Run your query against your local database and determine the number of records returned as described above. On the assignment page, turn in a text file `two_words.txt` that contains the number of records.

## Problem 2: Matrix Multiplication in SQL

Recall from lecture that a sparse matrix has many positions with a value of zero.

Systems designed to efficiently support sparse matrices look a lot like databases: They represent each cell as a record `(i,j,value)`.

The benefit is that you only need one record for every non-zero element of a matrix.

For example, the matrix

0 2 -1
1 0 0
0 0 -3
0 0 0

can be represented as a table

| row # | column # | value |
|---|---|---|
| 0 | 1 | 2 |
| 0 | 2 | -1 |
| 1 | 0 | 1 |
| 2 | 2 | -3 |

Take a minute to make sure you understand how to convert back and forth between these two representations.

Now, since you can represent a sparse matrix as a table, it's reasonable to consider whether you can express matrix multiplication as a SQL query and whether it makes sense to do so.

Within matrix.db, there are two matrices A and B represented as follows:

```
A(row_num, col_num, value)

B(row_num, col_num, value)
```

The matrix A and matrix B are both square matrices with 5 rows and 5 columns each.

**(g) multiply:** Express A X B as a SQL query, referring to the class lecture for hints.

What to turn in: On the assignment page, enter the value of the cell (2,3)

If you're wondering why this might be a good idea, consider that advanced databases execute queries in parallel automatically. So it can be quite efficient to process a very large sparse matrix --- millions of rows or columns --- in a database. But a word of warning: In a job interview, don't tell them you recommend implementing linear algebra in a database. You won't be wrong, but they won't understand databases as well as you now do, and therefore won't understand when and why this is a good idea. Just say you have done some experiments using databases for analytics, then mention the papers in the reading if they seem incredulous!

## Problem 3: Working with a Term-Document Matrix

The reuters dataset can be considered a *term-document matrix*, which is an important representation for text analytics.

The reuters dataset can be considered a *term-document matrix*, which is an important representation for text analytics.

Each row of the matrix is a *document vector*, with one column for every term in the entire corpus.

Naturally, some documents may not contain a given term, so this matrix is rather sparse. The value in each cell of the matrix is the term frequency. (You'd often want this this value to be a *weighted* term frequency, typically using "tf-idf": *term frequency - inverse document frequency*. But we'll stick with the raw frequency for now.)

What can you do with the term-document matrix *D*? One thing you can do is compute the similarity of documents. Just multiply the matrix with its own transpose $S = DD^T$, and you have an (unnormalized) measure of similarity.

The result is a square document-document matrix, where each cell represents the similarity. Here, similarity is pretty simple: if two documents both contain a term, then the score goes up by the product of the two term frequencies. This score is equivalent to the dot product of the two document vectors.

To normalize this score to the range 0-1 and to account for relative term frequencies, the *cosine similarity* is perhaps more useful. The cosine similarity is a measure of the angle between the two document vectors, normalized by magnitude. You just divide the dot product by the magnitude of the two vectors. However, we would need a power function (x^2, x^(1/2)) to compute the magnitude, and sqlite has built-in support for only very basic mathematical functions. It is not hard to extend sqlite to add what you need, but we won't be doing that in this assignment.

**(h) similarity matrix:** Write a query to compute the similarity matrix $DD^T$. (Hint: The transpose is trivial -- just join on columns to columns instead of columns to rows.) The query could take some time to run if you compute the entire result. But notice that you don't need to compute the similarity of both (doc1, doc2) and (doc2, doc1) -- they are the same, since similarity is symmetric. If you wish, you can avoid this wasted work by adding a condition of the form a.docid < b.docid to your query. (But the query still won't return immediately if you try to compute every result -- don't expect otherwise.)

What to turn in: On the assignment page, enter the similarity of the two documents '10080_txt_crude' and '17035_txt_earn'.

You can also use this similarity metric to implement some primitive search capabilities. Consider a keyword query that you might type into Google: It's a bag of words, just like a document (typically a keyword query will have far fewer terms than a document, but that's ok).

So if we can compute the similarity of two documents, we can compute the similarity of a query with a document. You can imagine taking the union of the keywords represented as a small set of (docid, term, count) tuples with the set of all documents in the corpus, then recomputing the similarity matrix and returning the top 10 highest scoring documents.

**(i) keyword search:** Find the best matching document to the keyword query "washington taxes treasury". You can add this set of keywords to the document corpus with a union of scalar queries:

```
SELECT * FROM frequency
UNION
SELECT 'q' as docid, 'washington' as term, 1 as count
UNION
SELECT 'q' as docid, 'taxes' as term, 1 as count
UNION
SELECT 'q' as docid, 'treasury' as term, 1 as count
```

Then, compute the similarity matrix again, but filter for only similarities involving the "query document": docid = 'q'. Consider creating a view of this new corpus to simplify things.

What to turn in: On the assignment page, enter the maximum similarity score between the keyword query and any document. Your SQL query should return a list of (docid, similarity) pairs, but you will submit only a single number: the highest score in the list.