SQL and Relational Algebra

This exrcise explores the Structured Query Languages using sqlite3 using the Command line (CLI) for a relatively large data file (10.7MB) in the cloud.

For most of these problems, the reuters.db database is used 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.

Problem 1: Inspecting the Reuters Dataset and Basic Relational Algebra

(a) A query that is equivalent to the following relational algebra expression-

$$\sigma_{docid} = 10398_{txt.earn}(frequency)$$

What is turned in: Run the query and determine the number of records returned and results stored in part_a.txt.

(b) An SQL statement that is equivalent to the following relational algebra expression-

$$\pi_{term}(\sigma_{docid=10398 txt.earn.and.count=1}(frequency))$$

What is turned in: Run the query and determine the number of records returned and results stored in part_b.txt.

(c) An SQL statement that is equivalent to the following relational algebra expression-

$$\pi_{term}(\sigma_{docid=10398 txt.earn.and.count=1}(frequency)) \; U \; \pi_{term}(\sigma_{docid=925 txt.earn.and.count=1}(frequency))$$

What is turned in: Run the query and determine the number of records returned and results stored in part_c.txt.

(d) An SQL statement to count the number of unique documents containing the word "law" or containing the word "legal" (If a document contains both law and legal, it is only counted once) *What is turned in: Run the query and determine the number of records returned and results stored in part_d.txt.*

(e) An SQL statement to find all documents that have more than 300 total terms, including duplicate terms. *What is turned in: Run the query and determine the number of records returned and results stored in part_e.txt.*

(f) An SQL statement to count the number of unique documents that contain both the word 'transactions' and the word 'world'. *What is turned in: Run the query and determine the number of records returned as described above and results stored in part_f.txt.*

Problem 2: Matrix Multiplication in SQL

Matrix multiply in a database might be a good idea- considering 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.

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

A (row_num, col_num, value) and, B(row_num, col_num, value)

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

(g) An SQL query to express A X B. *What is turned in: The value of cell (2,3) stored in part_g.txt.*

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.

Term-Document Matrix: Each row of the matrix is a document vector, with one column for every term in the entire corpus. 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. (Often this value is a weighted term frequency, ty/pically using "tf-idf": term frequency-inverse document frequency).

With the term-document matrix we can compute the similarity of documents by multiplying the matrix with its own transpose-

$$S = DD^T$$

and we will have an (un-normalized) measure of similarity.

The result is a square document-document matrix, where each cell represents the similarity. Here, similarity is defined as- 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. We 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 functions that we need.

(h) A query to compute the similarity matrix

$$DD^T$$

What is turned in: A file part_h.txt containing the similarity value of the two documents '10080_txt_crude' and '17035_txt_earn'.

We can also use this similarity metric to implement some primitive search capabilities. Considering a keyword query that one might type into Google, ty/pically a keyword query will have far fewer terms than a document.

So if we can compute the similarity of two documents, we can compute the similarity of a query with a document. We 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 re-computing the similarity matrix and returning the top 10 highest scoring documents.

(i) Find the best matching document to the keyword query "washington taxes treasury".

We then, compute the similarity matrix again, but filter for only similarities involving the "query document": docid = 'q'. *What is turned in: A file part_i.txt containing the maximum similarity score between the keyword query among all documents. The SQL query returns a list of (docid, similarity) pairs, but we are interested in the highest similarity score in the list.*