Analyzing Text Search Queries Utilizing Real Data: A Comparison of Oracle SQL and MongoDB

Lanae Blethen Kawa
Eastern Connecticut State University

Abstract

Document-based 'NoSQL' (not only SQL) databases have been rapidly gaining in popularity over relational databases in recent years, but they may not be the best fit for every use case. This study looks at a collection of metadata from PubMed cancer research articles that was populated into a MongoDB instance and an Oracle SQL database. Gene symbols and cell-line names were searched for in each database, and retrieval times as well as completeness of results sets were compared. The study was structured in this way to mimic a real-world use case where a researcher would utilize an academic search engine to find relevant articles. In this study, MongoDB had significantly faster retrieval times than Oracle SQL for a majority of the terms searched. Furthermore, this study also found that MongoDB has superior text searching capabilities when it comes to searching for non-language based strings that incorporate hyphen characters, as is the case with gene symbols and cell-line names. The author of this study concludes that MongoDB is the appropriate database technology for searching for terms within research article abstracts.

Introduction

Database Management Systems (DBMSs) have long been the cornerstone of nearly all forms of computing. After the Relational Database Management System (RDBMS) was introduced by Codd in 1970 [6], and Structured Query Language (SQL) developed by Boyce and Chamberlin in 1974 [5], the RDBMS-SQL paradigm became the dominant norm in computing, continuing into the present day.

However, the advent of Web 2.0 has created an explosion in the amount and type of data that is now available. It is typical for hundreds of thousands of users - or more - to require simultaneous access to a website's data [9]. Furthermore, much of this data is unstructured, which poses many challenges that the relational model is ill-suited to handle. RDBMSs are also very difficult to scale, owing to their reliance on expensive hardware if contained in a single server [15], as well as the difficulty in creating joins if distributed servers are needed to handle the load [12].

To meet the demands for better scalability and speed, as well as handling unstructured data, a group of databases began to emerge that are collectively known as 'Not Only SQL' (NoSQL) [16]. The "shared nothing" architecture common to NoSQL databases, where data is replicated across distributed servers, enables them to serve the large number of users common to today's applications [4]. Amazon's Dynamo and Google's Big Table were the forerunners in this new field, but as both of those systems were proprietary, they spawned a wave of open-source competitors to bring the benefits of NoSQL to the masses [4, 13]. Although this group of DBMSs is known under one moniker, there are in fact several distinct types, including column-oriented databases, graph data models, key-value stores, and document-based [13, 15, 16].

Perhaps the most well-known of these is the document-based MongoDB, developed by 10gen in 2009 [16].

With this plethora of distinct databases, it can be difficult to decide which one is the best fit for a given application or task, but there are a few general data characteristics that can help determine which system to choose. RDBMSs still play an important role in many businesses that require high precision and ACID (Atomicity, Consistency, Isolation, and Durability) guarantees for their data [4, 10, 12, 15, 16]. However, for businesses that deal in high volumes of data that isn't necessarily complex, or where absolute precision isn't necessary, the BASE (Basically Available, Soft state, Eventual consistency) guarantees provided by NoSQL databases are sufficient, especially given the speed boost and cost reduction associated with that type of technology [3, 4, 10, 16].

Aside from quality, type, and amount of data being utilized, the types of operations performed on a dataset are also a very important criterion for selecting the best fit database. There has been extensive research done comparing the performance metrics of NoSQL to SQL databases. Many of these analyses have focused on the basic Create, Read, Update, Delete (CRUD) functions that make up the bulk of database use.

Li and Manoharan found that document-based databases MongoDB and Couchbase were consistently fast performers for all types of operations, and NoSQL databases in general were much faster to instantiate than the Microsoft (MS) SQL Express database [13]. However, MS SQL Express did outperform other types of NoSQL databases in those same categories, and it was the overall fastest at fetching keys [13].

Aghi et al. also found that when they compared MongoDB with the popular open source MySQL RDBMS, insertion time was better for MongoDB [1], a finding backed-up by similar results in the paper by Gyorodi et al. [9]. MySQL was also much slower than MongoDB for update and delete operations by several factors [9]. As expected, MySQL performed better with highly structured data, but MongoDB excelled with non-relational data or datasets with large gaps and less structure [1]. MongoDB also outperforms MySQL when it comes to complex queries - document-based databases have no need to perform joins, which can save a significant amount of time as queries incorporate more attributes [1, 9].

Boicea et al. also assessed CRUD operations, but compared MongoDB to the RDBMS Oracle Database [3]. They also found that MongoDB outperformed Oracle in all the same areas as it did MySQL [3]. Additionally, they show that the performance gap between the SQL and NoSQL databases increasingly grows as the datasets upon which the operations are performed increase in size [3].

A major practical drawback to more widespread adoption of NoSQL solutions is their relative newness. They may not be a good fit for large organizations, as many of these open-source databases do not provide the robust support needed to be "enterprise-ready" [3]. The lack of a universal query language for NoSQL, akin to SQL, also introduces a steeper learning curve for businesses that wish to survey different kinds of NoSQL solutions [16].

So which is better, SQL or NoSQL? In this paper, the research described attempts to address that question by analyzing the outcome of building both types of databases for the same set of data and then running identical queries against each. The goal of this research was to determine whether MongoDB or Oracle SQL provide best performance for queries against a collection of PubMed article metadata with various filters such as locating articles mentioning a specific gene in the abstract. Current academic research work being undertaken by Dr. Garrett Dancik requires this functionality and it is the hope of this author that the research performed here may be able to assist Dr. Dancik in making a database tool selection for his work.

Based on current research in the field, it was this author's hypothesis that MongoDB would perform better given the size of the dataset, the loosely structured data, and the search engine type use case. The data indeed bore out this hypothesis, showing MongoDB to be have faster overall retrieval times than Oracle SQL for the majority of search terms. A more surprising finding was that MongoDB also returned results with higher qualitative accuracy due to the differences in how the two database languages construct text search indexes. In addition to speed and quality criteria, it is especially important to select a cost-efficient database in academic pursuits which often have limited funding, as the implementation and upkeep of databases can be quite costly, and picking the wrong technology could have significant negative impacts.

Materials and Methods

The specific data used in this experiment consists of academic genetic research article metadata obtained from scraping the PubMed online database, based on the work of Michael Underwood performed under the guidance of Dr. Garrett Dancik, which is available on GitHub [18]. In addition, a complete table of current genes was obtained from the European Bioinformatics Institute [7]. This data was originally populated into a free cloud-based MongoDB database offered by mLab [11] via Python scripts created by the author, one of which is shown in **Fig. 1**, and are available on GitHub [2]. However, due to difficulties in setting up a similar cloud-based platform for the SQL language, this was revised and the Python scripts were modified to populate a locally hosted MongoDB instance in order to control for differences in retrieval times that may be due to a network connection. The same data was then populated into a local Oracle SQL database, which like MongoDB is also free. The Oracle SQL Developer software provides the option to import data from a CSV file, so the text files used to populate the MongoDB collections were converted to CSV format and then imported into the appropriate tables in Oracle SQL.

A text index was built on the abstract attribute in each of the three MongoDB cancer collections and a 'context' index was built on the abstract attribute of the Article table in the Oracle database in order to ensure quickest lookup time. Queries where the gene symbol may match English words were originally structured to ensure that case sensitivity was applied to text search in order to retrieve only articles containing gene symbols and to eliminate articles where English words may match the symbol name. However, this parameter added additional overhead that other queries did not have, so those specific gene symbols were not considered for this experiment.

MongoDB queries were performed using the Mongo query language in the MongoDB command line shell. Oracle SQL queries were performed using the SQL language within Oracle SQL Developer. MongoDB comes with an internal set of diagnostic tools that can return information

```
from pymongo import MongoClient

def popDB(cancer):
        client = MongoClient('mongodb://lanaeBK:bioinfo4thewin@ds145667.mlab.com:45667/pubmed-data')
        db = client['pubmed-data']
        if cancer == 'bladder':
                collection = db.bladdercancer
        if cancer == 'lung':
                collection = db.lungcancer
        if cancer == 'pancreatic':
                collection = db.pancreaticcancer

        infile = open(cancer + '.txt')

        for line in infile:
                pmid = 0
                title = ''
                authors = ''
                source = ''
                abstract = ''

                for chunk in line.split('\t'):
                        description, content = chunk.split(':', 1)
                        if description == 'PMID':
                                pmid = int(content)
                        elif description == 'Title':
                                title = content
                        elif description == 'Authors':
                                authors = content
                        elif description == 'Source':
                                source = content
                        elif description == 'Abstract':
                                abstract = content
                                mongodoc = {"pmid" : pmid, "title" : title, "authors" : authors,
"source" : source, "abstract" : abstract}
                                collection.insert(mongodoc)

        client.close()

######
#MAIN#
######
popDB('bladder')
popDB('lung')
popDB('pancreatic')
```

Figure 1. *Python source code to translate PubMed metadata .txt files into MongoDB collections*

about performance time. Likewise, Oracle SQL Developer provides retrieval time information for each query.

All work was completed on a laptop running Windows 10 with an Intel® Core™ i7-4700MQ 2.40 GHz processor and 16 GB RAM. Microsoft Excel was used to track retrieval times and calculate mean, median, max, and min retrieval times. The MongoDB version used was 3.2.10. The Oracle SQL version used was 4.1.5, and the specific software used for database development was Oracle SQL Developer.

Results

The final number of database entries (called 'documents' in MongoDB) can be seen in **Table 1**. The same information for the equivalent Oracle SQL tables and rows can be found in **Table 2.** This information is an approximation as it was retrieved by using SQL query statements which use multiplication, division, and rounding, resulting in non-precise numbers.

Table 1. *Final collections in MongoDB database*

Collection	Documents	Size
bladdercancer	10,000	10.95 MB
lungcancer	10,000	10.94 MB
pancreaticcancer	10,000	10.71 MB

Table 2. *Final tables in Oracle SQL database*

Table	Rows	Size
Bladder Cancer Article	10,000	~23 MB
Lung Cancer Article	10,000	~23 MB
Pancreatic Cancer Article	10,000	~24 MB

The relational model used in the SQL database is shown in **Fig. 2** and an example of the schema used in MongoDB is shown in **Fig 3.**

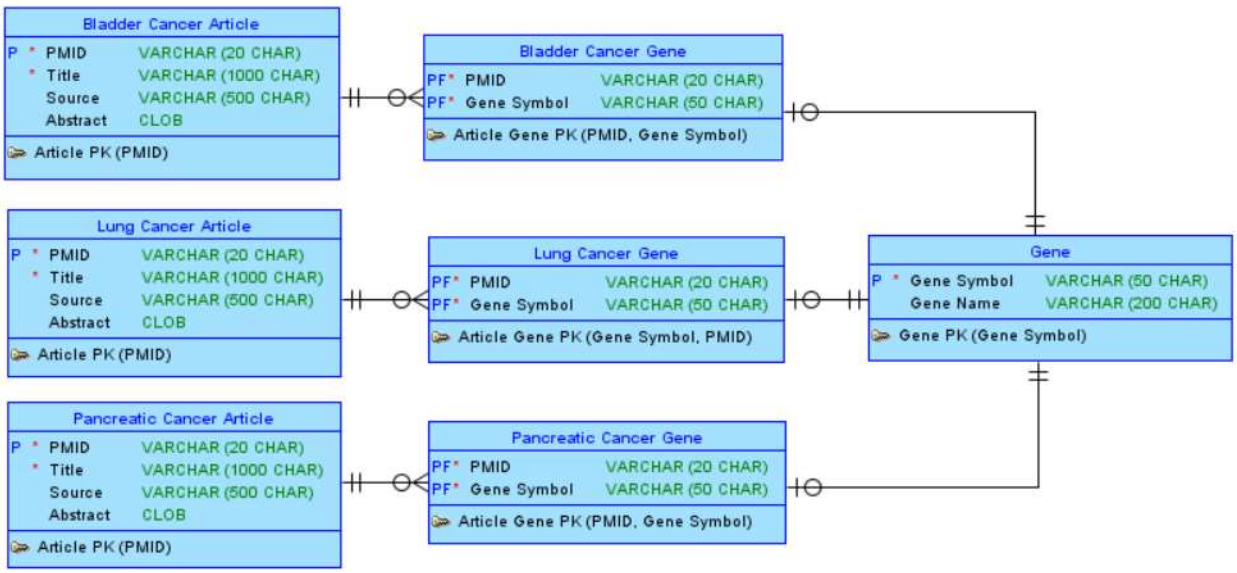

Figure 2. *Oracle SQL data model*

```
{
        "_id" : ObjectId("584f0886a0257949d0617c8e"),
        "title" : " Glucuronidation and UGT isozymes in bladder: new targets for
 the treatment of uroepithelial carcinomas?",
        "abstract" : " Bladder cancer has been linked to numerous toxins which c
an be concentrated in the bladder after being absorbed into the blood and filter
ed by the kidneys. Excessive carcinogenic load to the bladder urothelium may res
ult in the development of cancer. However, enzymes within the bladder can metabo
lize carcinogens into substrates that are safer. Importantly, these proteins, na
mely the UGT's (uridine 5'-diphospho-glucuronosyltransferases), have been shown
to possibly prevent bladder cancer. Also, studies have shown that the UGT1 expre
ssion is decreased in uroepithelial carcinomas, which may allow for the accumula
tion of carcinogens in the bladder. In this review, we discuss the UGT system an
d its' protective role against bladder cancer, UGT genetic mutations that modula
te risk from chemicals and environmental toxins, as well as targeting of the UGT
 enzymes by nuclear receptors.\n",
        "source" : " Oncotarget. 2016 Sep 27. doi: 10.18632/oncotarget.12277.",
        "authors" : " Sundararaghavan VL, Sindhwani P, Hinds TD Jr",
        "pmid" : 27690298
}
```


Figure 3. *MongoDB data schema*

For each cancer collection in MongoDB and table in Oracle SQL, a total of 21 search terms consisting of either the symbol of a specific gene or the name of a particular cell-line were each run five times against the collection, to return a count of articles that mentioned that gene in the abstract.

For instance, when submitting a query to search the bladder cancer collection for all articles mentioning the gene 'P53' the average time to return a count of all matching articles (201 in this example) is 7 ms in MongoDB and 9.6 ms in SQL. The query for MongoDB was constructed as follows:

```
db.bladdercancer.find(
{$text:
        {$search: "\"p53\""}
}).explain("executionStats"));
```

Similarly, in SQL the search query would be structured as follows:

```
select count(*) from  "Blddr Art"
where contains(abstract, 'p53', 1) > 0;
```

Table 3 shows the terms that were queried, the number *n* of matching articles that were found, and the average data retrieval time in milliseconds over five repeated queries for each collection in the MongoDB database. The same data for the SQL queries can be found in **Table 4.**

Table 3. *MongoDB query results*

Search Term	Bladder *n* returned	Bladder avg retrieval time (ms)	Pancreatic *n* returned	Pancreatic avg retrieval time (ms)	Lung *n* returned	Lung avg retrieval time (ms)
ACC-83	0	14.8	1	14.2	1	14.6
BRAF	12	0	31	1	83	2.8
BRCA1	7	0	48	1	21	0.2
BRCA2	6	0	59	3	18	0.2
ECV304	3	0	0	0	0	0.2
EGFR	91	3.8	142	6.8	888	32.2
FGFR2	9	0.2	4	0	10	0
HRAS	26	1	3	0	4	0
HSV-TK	4	0.2	1	0.2	1	0.2
KDM6A	9	0	3	0	1	0
KI-67	68	23.8	111	23.2	38	16.2
KRAS	22	1	337	12.4	249	8
L858R	0	0	0	0	102	3
MAP3K	0	0	0	0	1	0
MTOR	89	3	116	4	87	3
P53	201	7	160	5.4	179	6
RT112	37	1	0	0	0	0
IGF1R	7	0	5	0	11	0
UGT1	1	0	0	0	0	0
FGFR3	105	3	6	0	11	0
MIA PACA-2	0	147.2	122	138.8	6	131.6

Table 4. *Oracle SQL query results*

Search Term	Bladder *n* returned	Bladder avg retrieval time (ms)	Pancreatic *n* returned	Pancreatic avg retrieval time (ms)	Lung *n* returned	Lung avg retrieval time (ms)
ACC-83	3	7.4	19	20.8	10	21.8
BRAF	12	8.2	31	9.4	83	8
BRCA1	7	7.6	48	6.8	21	7.4
BRCA2	6	7.2	59	7.4	18	7
ECV304	3	8.4	0	7.8	0	7.4
EGFR	91	8	141	8	887	8
FGFR2	9	7.6	4	8	10	7.8
HRAS	26	7.8	3	8.2	4	8.2
HSV-TK	4	8.2	2	7.6	0	7.2
KDM6A	9	9.2	3	12.2	1	9.4
KI-67	69	9.4	120	9.6	43	6.6
KRAS	22	10.2	337	1	249	12.2
L858R	0	4.2	0	0	102	10.2
MAP3K	0	9.6	0	6.4	1	3.6
MTOR	89	3	116	7	87	6.2
P53	201	9.6	160	0.4	179	6.2
RT112	37	4.2	0	7.4	0	3.2
IGF1R	7	7.4	5	9.2	11	6.2
UGT1	1	6.4	0	9.4	0	9.2
FGFR3	105	9.2	6	6.4	11	4.2
MIA PACA-2	0	10.2	98	15.2	5	13

A comparison between MongoDB and Oracle SQL of the mean and medium retrieval times for each cancer collection/table (excluding the search term ‘mia paca-2’, which will be discussed later) can be found in **Fig. 4, Fig. 5, Fig. 6, Fig. 7, Fig. 8, and Fig. 9.**

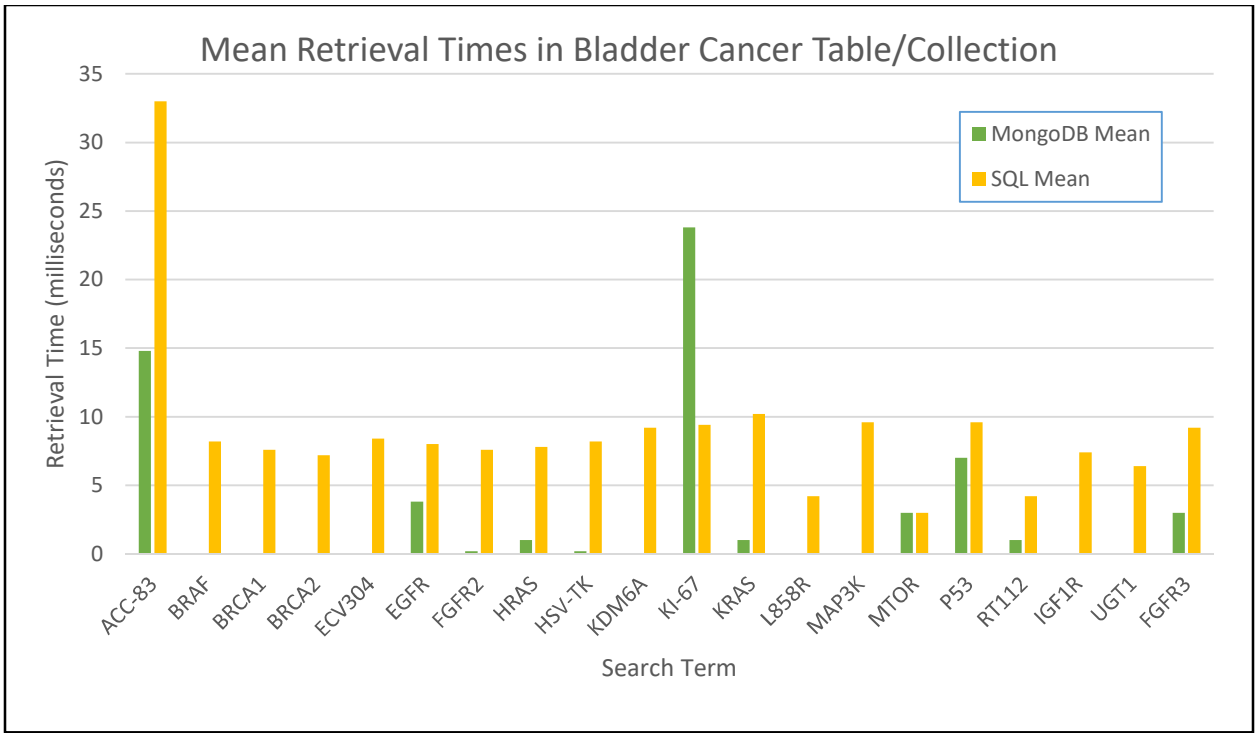

Figure 4. *Mean retrieval times for bladder cancer article collection (MongoDB) and table (SQL)*

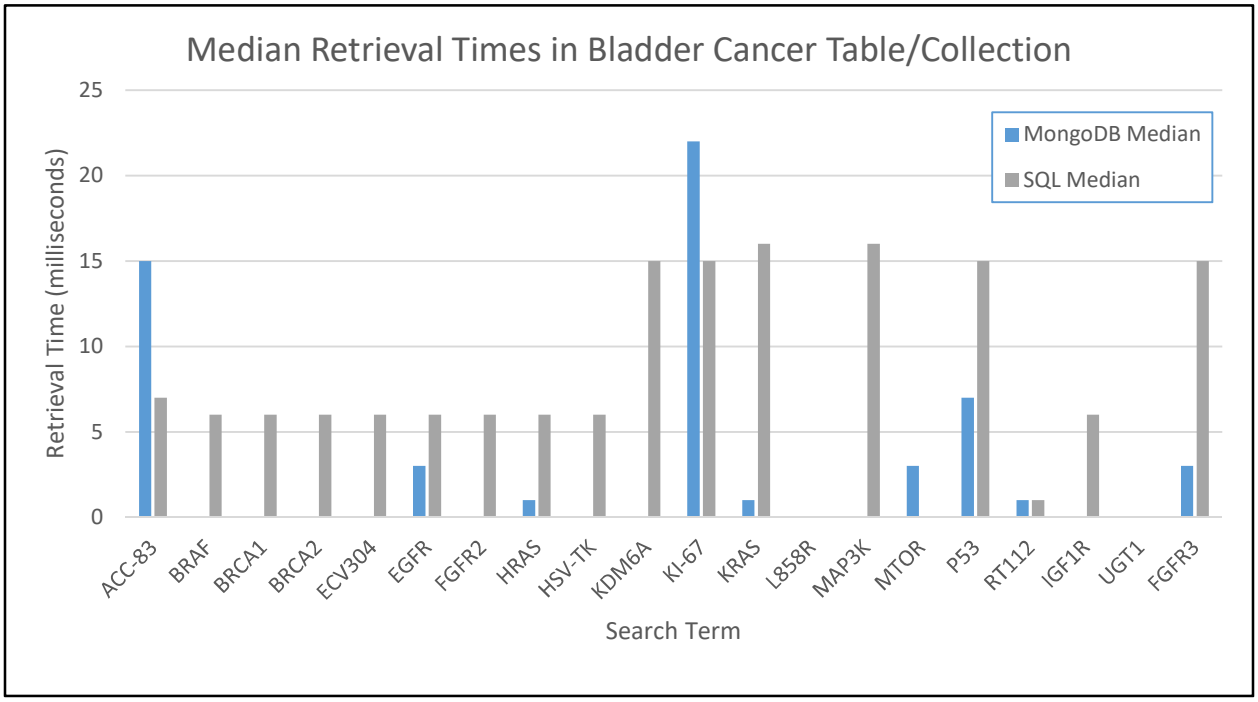

Figure 5. *Median retrieval times for bladder cancer article collection (MongoDB) and table (SQL)*

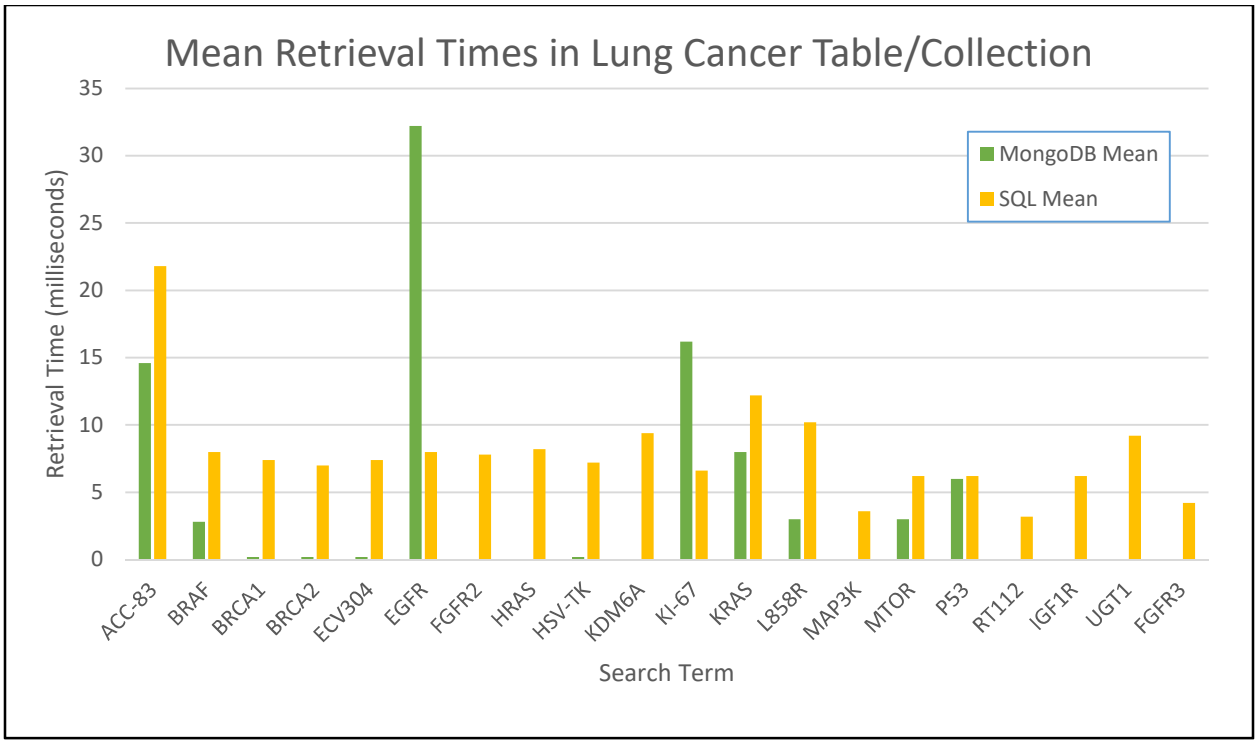

Figure 6. *Mean retrieval times for lung cancer article collection (MongoDB) and table (SQL)*

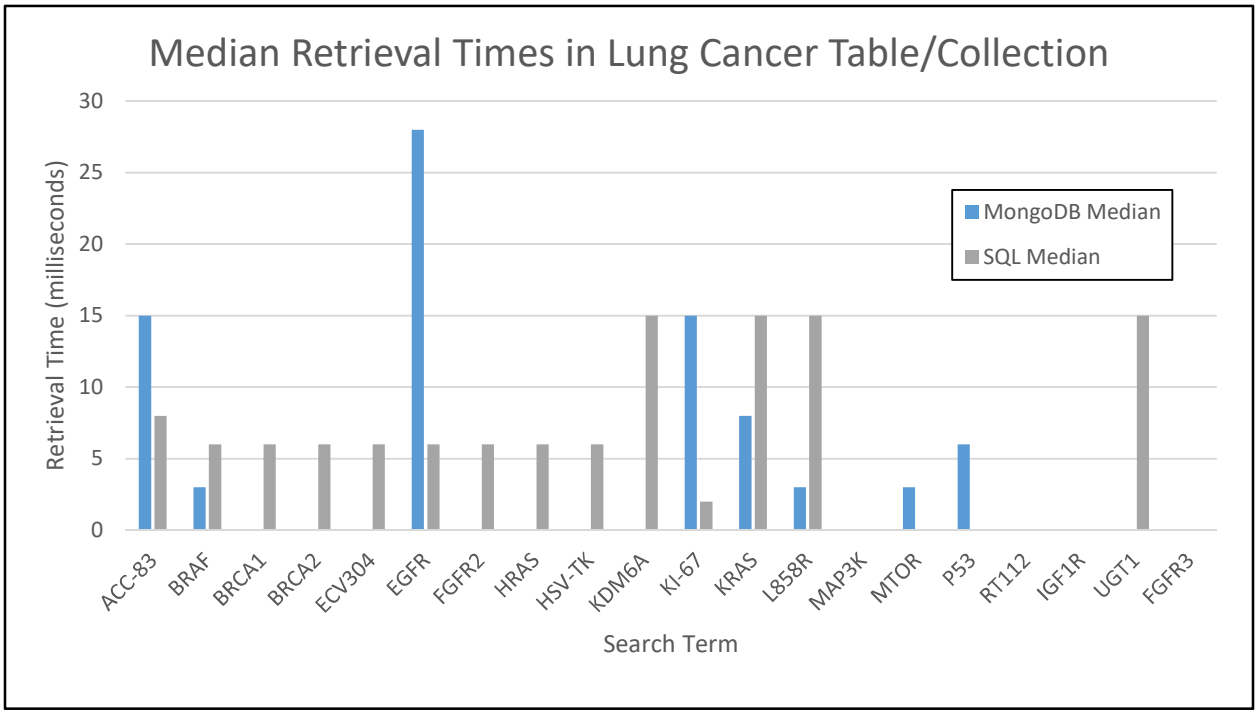

Figure 7. *Median retrieval times for lung cancer article collection (MongoDB) and table (SQL)*

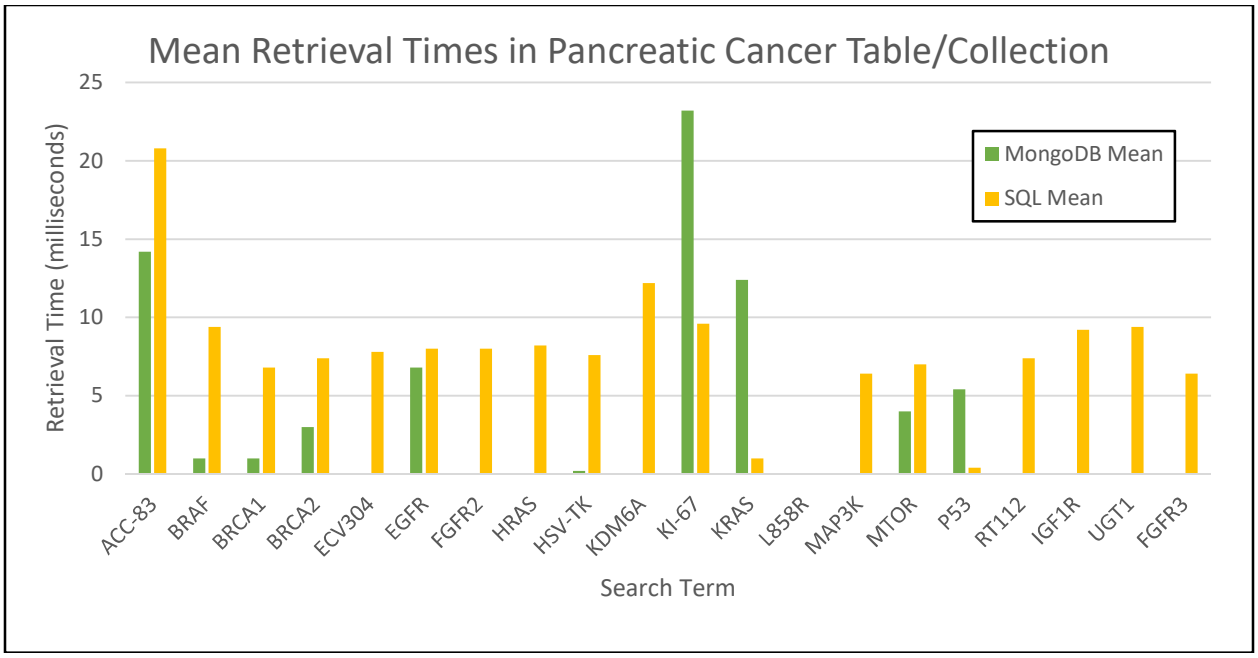

Figure 8. *Mean retrieval times for pancreatic cancer article collection (MongoDB) and table (SQL)*

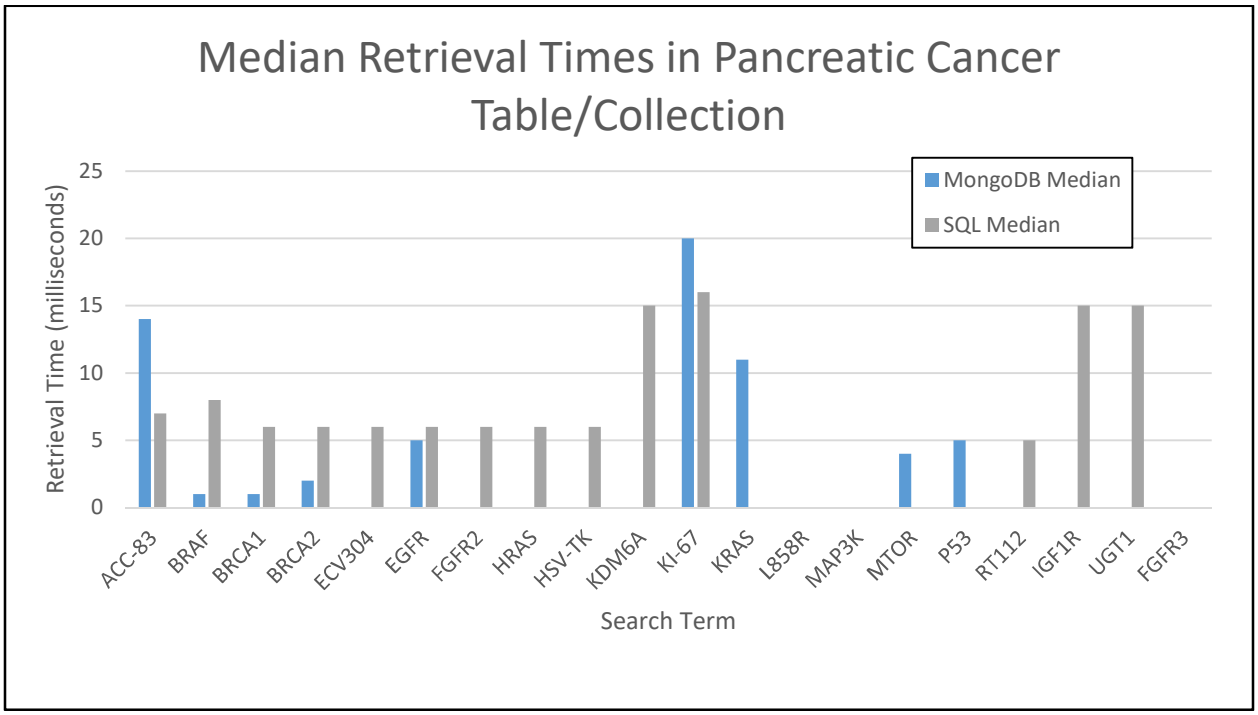

Figure 9. *Median retrieval times for pancreatic cancer article collection (MongoDB) and table (SQL)*

Discussion

A surprising finding from this research was the difference in text search capabilities between the two types of databases. It was simple to add a text index to each of the MongoDB collections. The index for Oracle SQL was exponentially more difficult to build. Oracle SQL offers three types of text search indexes: context, ctxcat, and ctxrule [17]. I initially built a 'context' index, but during preliminary queries noticed that MongoDB was successfully retrieving article abstracts that contained hyphenated terms like 'acc-83' while SQL was not. This is evident in the different number of results returned for searches of the same term in a collection or table consisting of the same article data, as indicated by the red font in **Table 4**.

Upon further perusal of the Oracle documentation, I discovered that the 'context' index separates words when it comes across a hyphen, which it regards as a 'stopjoin', and also uses the hyphen as a 'minus' in search queries. For instance, searching for 'acc-83' in SQL will look for the term 'acc' *less* the term '83' [17]. This is because the 'context' index is optimized for searching for actual words in languages like English, French, and Spanish, and the terms used in this experiment are not actual words but rather names of genes and cell-lines. The 'ctxrule' index doesn't fit this use case because terms must match exactly, which is not helpful in this case where a single, short string needs to be found in an abstract that may consist of more than 4,000 characters [17]. Furthermore, due to the length of some of these records' abstracts, when constructing the SQL data model, that field must be designated as a 'CLOB' (character large object), due to the limits of both the TEXT datatype (no more than 2,000 bytes) and the VARCHAR2 datatype (no more than 4,000 bytes) [17]. This means that the 'ctxcat' index could also not be used, as it only works on TEXT or VARCHAR2 datatypes [17].

After discovering that the only available text index was 'context', I attempted to modify the index to include the hyphen as part of the search term. In Oracle SQL, this is done by constructing a custom 'lexer' to include a 'printjoin' (the character to include in the search) and adding it as a parameter to the 'context' index [17]. However, after I successfully altered database permissions, constructed the parameter and rebuilt the index, the subsequent search results were even more skewed than the original index that lacked the 'printjoin' parameter. I ended up dropping this new index and rebuilding the index the way it was originally constructed even though I knew not all of the results would be correct. This is indicated in **Table 4**, where the red font color indicates where the number of articles returned in Oracle SQL differed from the number returned by MongoDB for the same search term and collection. I believe that this alone disqualifies SQL from being suitable for this use case, as it misses articles that contain hyphenated search terms, and returns incorrect articles that contain one part of the hyphenated search term.

Setting all that aside for a moment, it is also clearly evident from the bar graphs that whether median retrieval time or mean retrieval time are being compared, MongoDB is consistently faster than SQL in for the majority of non-hyphenated search terms. For instance, looking at the median retrieval times in the pancreatic article collection, shown in **Fig. 9**, Oracle SQL was faster than MongoDB only 25% of the time. It is not immediately evident why SQL was faster in these cases. It may be a result of how the indexes were built, or could simply be caused by fluctuations in machine resources at the instant of the query.

As shown in the MongoDB results in **Table 3**, it was interesting to note that the gene search for MIA PACA-2 returned no article results from the Bladder collection and yet had a search time of 147.2 ms while the search for the gene L858R had 0 ms search times when returning 0 results, and only 3 ms search time when returning 102 records from the Lung collection. It is possible that this is because the text search runs more slowly with hyphenated gene symbols, although the search for gene HSV-TK did not behave similarly, so there may be other factors at play, such as the fact that MIA PACA-2 is the only term that has a space in it, so traversal time of the index is probably longer as a result.

The creation, population, querying, and especially the index building for the MongoDB database was subjectively much simpler than the setup of the Oracle SQL database. This author does have moderate experience with MongoDB and almost none with Oracle SQL, which could be the reason for this discrepancy in "ease of use". However, it does appear that MongoDB has a simpler, more "user-friendly" implementation, which could be an important consideration in selecting a database that will be used in an academic environment and likely maintained by students or volunteers who may have limited database experience. I also feel that SQL has a significantly steeper learning curve with a vast amount of more esoteric commands and rules to learn compared to MongoDB, which is relatively easy for someone with no prior database experience to pick up.

The datasets of 10,000 articles per cancer type may have been too small to obtain meaningful and robust data. In the future, it would be useful to use a set of 100,000 or more articles to search through in order to get a sense of how MongoDB and Oracle SQL compare with each other at larger scales and determine if the results achieved by Boicea et al. for basic CRUD operations also hold true for this type of more complex querying [3]. Furthermore, a larger sample of search term may show different results, as there is no obvious reason why SQL retrieval times were faster than MongoDB for a handful of non-hyphenated terms.

In addition to query response time and completeness of result sets, another factor to assess in deciding which of these databases is a better fit for this dataset is whether the data needs to meet ACID quality standards [3, 10, 12, 15, 16]. In this case, with the data being scraped from a website, it is by nature not a highly structured source of data; some fields may be missing and the fields will also have widely varying lengths. The way the data would be queried is also more informal and ad hoc, as it simply a search engine type use case. This indicates that BASE standards [3, 4, 10, 16] are satisfactory for this type of data, and therefore SQL's more expensive ACID guarantees are not required. As new data is acquired and added to the database, it will also be much easier to add to a MongoDB database than a SQL database. Thus, based on this research, it is recommended that MongoDB be selected for this specific use case of data querying.

References

[1] R. Aghi et al., "A comprehensive comparison of SQL and MongoDB databases," in *International Journal of Scientific and Research Publications*, vol. 5, no. 2, Feb. 2015.

[2] L. Blethen Kawa. "damakawa/sql-vs-nosql-research," *GitHub*. [Online]. Available: https://github.com/ damakawa/sql-vs-nosql-research.

[3] A. Boicea et al., "MongoDB vs Oracle – database comparison," in *Emerging Intelligent Data and Web Technologies (EIDWT), 2012 Third International Conference*, Sept. 2012, pp. 330 –335.

[4] R. Cattell, "Scalable SQL and NoSQL data stores," in *Acm Sigmod Record*, vol. 39, no. 4, Dec. 2011, pp. 12-27.

[5] D. D. Chamberlin and R. F. Boyce, "SEQUEL: A Structured English Query Language," in *Proceedings of the 1974 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, New York, NY, USA, 1974, pp. 249–264.

[6] E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," *Commun. ACM*, vol. 13, no. 6, pp. 377–387, Jun. 1970.

[7] European Bioinformatics Institute. [Online]. Available: ftp://ftp.ebi.ac.uk/pub/databases/genenames/new/tsv/hgnc_complete_set.txt. [Accessed: 17-Nov-2016].

[8] A. Floratou et al., "Can the Elephants Handle the NoSQL Onslaught?" in *Proceedings of the VLDB Endowment*, Istanbul, Tukey, vol. 5, no. 12, Aug. 2012, pp. 1712-1723.

[9] C. Gyorodi et al., "A Comparative Study: MongoDB vs. MySQL", in *The 13th International Conference on Engineering of Modern Electrical Systems*, Oradea, Romania, June 2015.

[10] N. Jatana et al., "A Survey and Comparison of Relational and Non-Relational Database," in *International Journal of Engineering Research and Technology*, vol. 1, no. 6, Aug. 2012.

[11] mLab. [Online] Available: https://mlab.com.

[12] N. Leavitt, "Will NoSQL databases live up to their promise?" in *Computer*, vol. 43, no. 2, Feb. 2010, pp. 12-14.

[13] Y. Li and S. Manoharan, "A performance comparison of SQL and NoSQL databases," in *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, 2013, pp. 15–19.

[14] P. Milic, "Comparative analysis of performance of open source databases in web application development," in *MIT-2016,* Vrnjacka Banja (Serbia), Budva (Montenegro), 2016, pp. 1-8.

[15] C. Nance et al., "Nosql vs rdbms - why there is room for both," in *Proceedings of the Southern Association for Information Systems Conference*, Mar. 2013, pp. 111-116.

[16] A. Nayak et al., "Type of NoSQL Databases and its Comparison with Relational Databases," in *International Journal of Applied Information Systems*, vol. 5, no. 4, Mar. 2013, pp. 16-19.

[17] Oracle. 'Database Reference'. [Online] Available: http://docs.oracle.com/cd/B28359_01/server.111/b28320/toc.htm [Accessed: 15-Dec-2016].

[18] M. Underwood. "mjunderwood/minePM," *GitHub*. [Online]. Available: https://github.com/mjunderwood/minePM. [Accessed: 10-Oct-2016]