A Comparison of Search Engine Technologies for a Clinical Data Warehouse

Georg Dietrich, Georg Fette and Frank Puppe
University of Würzburg and DZHI (Deutsches Zentrum für Herzinsuffizienz)
{dietrich, fette, puppe}@informatik.uni-wuerzburg.de

Abstract

A clinical data warehouse (DW) can be used to recruit patients for clinical studies or statistical analysis. For improved user experience, it is crucial that the search engine technology of the DW answers user queries quickly. In this paper, we investigate the performance of the two most popular technologies for regarding structured and unstructured data query answering: a database and a search engine. Our empiric results show that search engines have advantages for complex queries.

1 Introduction

A clinical data warehouse makes data available for a variety purposes, e.g., information retrieval and statistical evaluations. The data consists of basic data, symptoms, diagnoses and therapies. Use-cases are the retrieval of patients for clinical studies, which have several inclusion and exclusion criteria, statistical analysis of frequencies of patient groups, the search for risk factors for specific diseases and statistical quality checks. For efficient usage, quick query answering is crucial. In this direction, we compare the performance of two alternative techniques in a real world application featuring a clinical data warehouse DWH utilized at the University of Würzburg.

Currently (June 2014) the data warehouse of the University Hospital of Würzburg consists of basic data, diagnoses, laboratory findings and echocardiography data for the years 2012 and 2013. There are about 70 0000 cases with more than 25 million facts available. To protect the privacy, all data has been pseudonymised. The mapping of the pseudoids to the patient-ids is managed by a third party, which can approve applications to e.g. recruit patients for studies with the data warehouse.

In order to work with the data warehouse, it must be able to answer queries quickly. Therefore it is necessary to store the data in an efficient way. Furthermore fast access and intuitive usability are important.

There are several ways to design such an information retrieval system. A basic approach is to use a database management system. As a first step, a schema has to be designed. This is a non-trivial task, because the knowledge base consists of about 55 000 concepts like the laboratory finding of natrium, the age of a patient or the diagnosis heart attack. After that, indices need to be created for the tables to speed up the system. Finally, the algorithms, which automatically create queries for the database, have to be implemented.

Another approach is to use a Resource Description Framework (RDF). Here, the schema has to be specified first, too. The 25 million facts are then stored as RDF triples, such as patient X has laboratory finding of natrium of 140 mmol/l. In this example, patient X is the subject, a laboratory finding of natrium is the predicate and 140 mmol/l is the object. The RDF data model can be queried with the "SPARQL Protocol And RDF Query Language" [6].

The user queries contain usually about ten or more parameters. This is quite a lot and the RDF storage didn't scale for our challenges.

The third approach is to use a search engine. We used Apache Solr [2], which needs a schema for the documents and their fields. This is similar to the database schema, being flexible to new fields and changes.

Our data warehouse query should provide the following features:

- An intuitive usable graphical user interface to create easily queries, whose result is displayed in a clear way.
- A search for hierarchical structures like a diagnosis-tree.
- Span and segment queries to search medical concepts, which consist of several words.
- The system is able to use synonyms and abbreviations for medical query terms.
- Very fast response time for complex user queries.

2 Background

The clinical data consists of four data types:

- Numeric Values: Most Laboratory Findings are floats with a few decimal places like haemoglobin = 16.58 g/d
- Boolean Values: Diagnoses are represented as boolean values. When a disease is diagnosed it is stored like hypertension = true
- Text Values: Several medical reports of findings exist as texts like discharge letters or electrocardiogram reports.
- Enumerations: Many Attributes have a view values like sex (female, male) or type of treatment (residential, semi-residential, ambulant)

2.1 Data Schema

At the first approach all available facts were stored in a relational database. A simplified model of the data schema with two basic tables is shown in Figure 1.

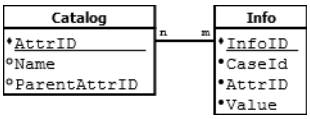

Figure 1: Simplified relational model of the database with a triple structure in the Info-table: CaseID, AttrID, Value

All attributes, which a patient can have, are stored in the *Catalog*-table, which also represents hierarchical relations. Examples for attributes are natrium (lab data), sex (basic data) or I50.22 Chronic systolic (congestive) heart failure (diagnostic data). The values for the attributes are stored in the *Info*-table: The CaseID represents one ”case” of a patient (including admission and discharge dates) including all data for that patient in that time period. The CaseID, AttrID and the Value form a triple structure: For one case and one attribute exists one value, e.g. the patient of a case has blood pressure of 125. It is possible that one attribute has several values, like multiple measurements of one attribute at different time stamps. This is mapped with several rows in the table.

An alternative schema with one big table and a column for every attribute was tested, but discarded, because only ca. 1000 columns per table were allowed (just for the diagnoses, we needed more than 16000 columns).

2.2 Hierarchical search

A special function is the hierarchical search. Our terminology is hierarchically ordered, like the ICD-10 coded diagnoses [1]. The international classification of diseases is a catalog for epidemiology, health management and clinical purposes. It is hierarchically structured as you can see in Figure 3.

If a specific disease like *I20.0 Instablile Angina petoris* is diagnosed, then the ”parent”-disease (here: *I20. Angina petoris*) exists, too. Usally, a very specific diagnosis like *I20.0* is documented, which has a high depth in the catalog. But the data warehouse user may search for a more general diagnosis like *I20*. To meet this requirement, new facts were generated by preprocessing, i.e. setting all parent diagnoses of a diagnosis ”true” thus propagating the diagnosis up in the tree.

2.3 Graphical user interface

The graphical user interface consists mainly of three views. In the catalog view (Figure 2 and 3) all attributes are hierarchically sorted. After every attribute name the total number of occurrences in the data warehouse is shown. Attributes can be dragged from the catalog-view (Figure 3) and dropped in the query view (Figure 4). Operators and constraints can be applied to this attributes in the query view, e.g. numeric range selection. If one attribute has more than one value in one case, it is possible to specify which one should be selected (first, last, min, max). Moreover the boolean operatores ”And”, ”OR” and ”Not” are available for combinations. In the result view the query matching cases are displayed tabularly (Figure 5).

3 Evaluation: A speed-test between Solr and a DBMS

A database server and a search platform were tested as storage engine for the data warehouse application.

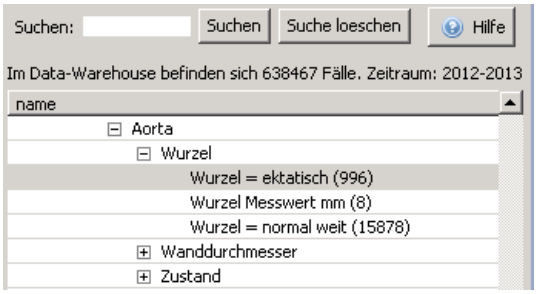

Figure 2: The catalog view (in german) of the data warehouse shows hierarchically structured all queryable attributes and their number of occurrences.

Figure 3: An example of the hierarical structure of the ICD-10 catalog is in the catalog view.

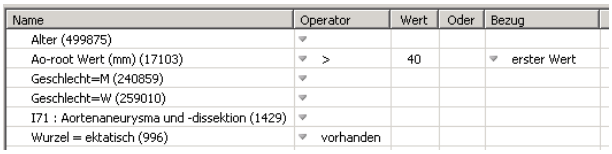

Figure 4: In the query view of the data warehouse properties of the attributes can be set.

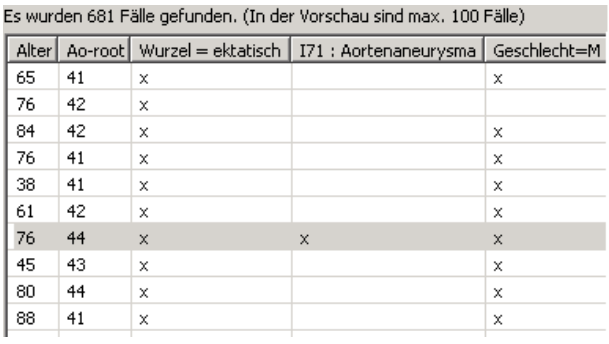

Figure 5: In the result view of the data warehouse are query hits shown in a tabular style.

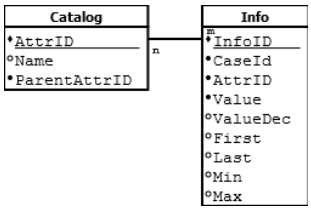

Figure 6: The relational model of the database with the triple structure in the Info-table and additional flags for the first, last, min or max value of one attribute for one case.

3.1 Setting

For this test, various queries have been made to the systems and the response time was measured. The database system is a Microsoft SQL Server [5] and the search platform is Apache Solr 4.8 [2]. The database schema is shown in figure 6. It has been extended to the first example 1 with the columns ValueDec, which is a decimal(8,2) column, and the four columns (first, last, min and max), which can have the values "1" or null. Because some attributes can have more than one value in a case, these four columns mark, if the current is e.g. the first occurence in the case. Strings values are stored in the normal value-field and numbers are stored in the decimal-field for a quicker access. The Info-table has in the first runs, shown in Table 1, a small index on the two columns CaseID and AttrID. In the last test-runs, shown in Table 2, the index is extended to the columns CaseID, AttrID, ValueDec and First, Last, Min, Max. In our test, the database did not use caching.

Solr has a document centered approach, so all facts of one case are pooled into one document and these documents are then indexed. The Solr-schema consists of dynamic fields for every attribute, which means, that every attribute has got its own index.

In our application, the first 100 hits and the total count of hits are displayed. For these two information are two requests necessary in the database, a top-100-query and a count(*)-query, Solr provides these two information in one query response.

14 settings were tested, three with boolean attributes, eight with numeric attributes and three with a text field. For every setting five queries have been send to the server and the response time has been measured. In Table 1, the average values of every five queries are shown in milliseconds.

Several diagnoses-attributes were used for the queries with the boolean-values. The average occurrence of a attribute was about 30 000 times, but some attributes had a occurrence of a few thousand, others had up to 100 000 occurrences. For the numeric tests, laboratory findings, which had about 100 000 occurrences, were queried. If a condition was applied to a numeric value, it was always a range query with a lower- and a upper-bound. 25 000 texts were used for the word-queries, which were realized with the like-operator. In the first word-test a single word was requested, in the second a word with the wildcard * and in the third test three AND-connected words were tested.

3.2 Results

Overall, it has been found, that a full indexed database is faster Solr, except the DB must join tables, than Solr is faster. If the db isn't full indexed, Solr is always faster. As it is show in Table 1 and 2 the database is only faster, if one attribute was queried and the index covered all used columns. The query for one boolean attribute is on the DB fast, because for a diagnosis query the value column does not need to be checked, because only positive records are stored in the database. So the query can be answered by only using the columns AttrID and CaseID, witch are contained in the small index of the table and this is very effective.

	db top 100	db count	db sum	Solr
1 Bool	40	32	73	115
3 Bool And	2267	140	2407	48
3 Bool OR	6465	143	6608	235
1 Word	18	3280	3248	218
1 Word *	2332	2399	4731	155
3 Words	39	6579	6645	445

Table 1: Response time in milliseconds for various queries. A comparison between a MS SQL DB and Apache Solr for querying boolean values or words in text fields. The boolean attributes were ORed and ANDed.

But this does not work for the numeric queries, because it must be checked for every record if the value was in the selected range or if the flag was set in the First, Last, Min, Max flied. In Table 2 the response times are shown for the small and the extended index for the DB. In the small index not all columns are included, which are required to answer the query. In contrast, the extended index contains all relevant columns. As you can see in Table 2, there is a big difference in the respond time, if the DB can use an index or it can't. Solr can use its index on the numeric field to answer quickly, too.

If more than one attribute is queried, the database must join the Info-table with itself, because the facts are stored in a triple structure in the database. This is quite expensive and it explains, why Solr is faster, when more than one attribute is queried. Even the index of the DB doesn't help, which can be seen in the tests with three boolean or numeric attributes.

The database is quite fast with fetching the first 100 results for a single or a multiple word-query, but it is quite slow for a word-*-expression. The DB does not have to join tables to answer the 3-word-query, but the respond time is twice as long. Solr is much faster for text queries, because the texts are indexed here and in the DB they are not indexed.

It can be also observed, that the database is much slower, when the attributes are ORed and not ANDed.

But the main finding is, that Solr is on average drastically faster than the database system. It looks like, Solr doesn't take significant longer, if more attributes were queried.

If the tests are considered, where all necessary data was indexed, Solr is a bit slower, if one attribute was requested only, but if three attributes were queried, Solr is nearly ten times faster than the DB.

4 Additional Features of the Search Engine

By using the search engine, some new text query features are now possible.

Segment and span search

Text fields can be efficiently searched. It is not only possible to search by multiple terms, it is also possible to make span queries. A span query can be used to find multiple

	DB with small index			DB with extended index			Solr
	top 100	count	sum	top 100	count	sum	
1 Num.	692	3 423	4 115	4	50	54	167
1 Num. with cond.	403	966	1 369	5	184	189	242
3 Num. with cond. AND	590	1 904	2 494	52	253	305	136
3 Num. with cond. OR	10 438	5 086	15 524	7 207	378	758	234

Table 2: Response time comparison between a MS SQL DB and Apache Solr for querying numeric values with and without conditions. The attributes were ORed and ANDed. All results are in milliseconds. The small DB-index does not contain all required columns, the extended index contains all.

terms near each other, without requiring the terms to appear in a specified order. This can be a powerful tool for searching concepts, which consist of several words, like heart failure. Consider following sentence:

(1) Heart: left ventricular failure.

It is possible to set the maximum distance, the two terms may be away from each other. The words *heart* and *failure* have a distance of three words, so this technique works well here. But it won't work in the next tow examples:

(2) Kidney: renal failure. Heart: normal after transplantation.

(3) Heart: sinus rhythm, normal large left ventricle, aortic root normal width, right ventricular failure.

In example 2, the context of *failure* is *kidney* not *heart*. While this can be covered in the tool by determining an order for the terms, the span query approach is not suitable for the third example. The distance of the two terms is too far and the context can not be safely resolved.

Therefore, another approach was implemented: The segment search. Many text documents are structured like the examples above. One text consists of an enumeration of concepts like heart or kidney. Every concept is followed by a colon and list of findings. Therefore, it make sense to split these texts in segments, like in example 1 and 3. Example 2 would be split in two segments. This procedure is a preprocessing step, which makes it possible to search in these segments quickly. A query searches only in individual segments and doesn't mix them up. So, if you query example 2 with the terms *heart* and *failure*, you will get no hit.

Synonym search

Another feature of the search engine implementation is, that queries are complemented with synonyms. Every term of a query is analyzed if it is a medical term, which has synonyms or abbreviations, these terms are added to the query. With this feature, a higher recall can be achieved.

5 Related Work

An empirical study on performance comparison of Lucene and a relational database has been made by Jing [4]. Apache Solr uses the Apache Lucene search library for building and querying the index. Jing tests a MS SQL Server, too, but with a full-text-index. Unfortunately, this feature was not available to us. Furthermore they query only one table and they don't make any joins. But their results also say, that Lucene is faster than an unindexed database. Except, if combinational queries, with more than one where-clause, which could be ORed or ANDed, were tested, Lucene was on average quicker. Jing uses synthetic generated data and tested only queries without join operations, we had real data with many joins.

The Léon Bérard Cancer Center in France [3] implemented their information retrieval systems also with Solr, but only as full-text search engine and not for structured data.

6 Conclusions

In this paper, we presented a brief overview over the main functions and the GUI of our clinical data warehouse query tool. We described the setting for our storage engines and our requirements. We showed and explained in several tests the advantages and disadvantages of relational database and Solr for query answering. It has been found, that Solr is faster than an unindexed database. If the DB was full indxed, then it is faster as Solr, except the DB must join tables. In that case, Solr is faster again.

References

[1] World Health Organization (WHO) : International Classification of Diseases (ICD). `http://www.who.int/classifications/icd/en/` (2014), [Online; accessed 20-June-2014]

[2] Apache Software Foundation: Apache Solr. `http://lucene.apache.org/solr/` (2014), [Online; accessed 20-June-2014]

[3] Biron, P., Metzger, M.H., Pezet, C., Sebban, C., Barthuet, E., Durand, T.: An information retrieval system for computerized patient records in the context of a daily hospital practice: the example of the léon bérard cancer center (france). Applied clinical informatics 5(1), 191–205 (2014)

[4] Jing, Y., Zhang, C., Wang, X.: An empirical study on performance comparison of lucene and relational database. Communication Software and Networks, International Conference on 0, 336–340 (2009)

[5] Microsoft: Microsoft SQL Server. `http://msdn.microsoft.com/en-us/library/bb545450.aspx` (2014), [Online; accessed 20-June-2014]

[6] W3C: W3C, SPARQL 1.1 Protocol. `http://www.w3.org/TR/sparql11-protocol/` (2014), [Online; accessed 20-June-2014]