Supporting Ontology-based Keyword Search over Medical Databases

Anastasios Kementsietsidis, Ph.D. Lipyeow Lim, Ph.D. Min Wang, Ph.D.
IBM T.J. Watson Research Center, 19 Skyline Drive, Hawthorne, NY 10532, USA.

The proliferation of medical terms poses a number of challenges in the sharing of medical information among different stakeholders. Ontologies are commonly used to establish relationships between different terms, yet their role in querying has not been investigated in detail. In this paper, we study the problem of supporting ontology-based keyword search queries on a database of electronic medical records. We present several approaches to support this type of queries, study the advantages and limitations of each approach, and summarize the lessons learned as best practices.

1. Introduction

The proliferation of medical terms is a major obstacle in the sharing of medical information among different shareholders (e.g., hospitals, clinicians, pharmaceutical companies etc.). Even different clinicians within a hospital often use distinct terms to refer to the same diagnosis, while symptoms are often recorded to a patient's record in varying levels of granularity. For example, one clinician might describe a patient diagnosis using the term *"Pineoblastoma"*, while another might use the (synonym) term *"PNET of Pineal Gland"*. In the patient's record a generic term like "Brain Neoplasm" might be recorded instead of the more specific *"Pineoblastoma"* (where the latter term is said to be a *hyponym* of the former). Coding systems such as SNOMED or ICD9, and ontologies like the National Cancer Institute (NCI) Thesaurus, encode terms and their relationships, yet they do not prevent different clinicians from using hyponyms, hypernyms, or synonyms in an electronic medical record (EMR).

EMRs are usually stored for efficiency in relational databases and one would expect that it is straightforward to bridge the gap between the term ontology and the EMR database so as to use the former to retrieve records from the latter. The following example illustrates that unfortunately this is not the case. Consider an EMR database, like the one in Figure 1, which stores hospital patient visits. For each visit, it stores its identifier *"vid"*, the *"date"*, patient identifier *"patID"*, and a diagnosis *"diag"* using terms from the NCI Thesaurus. Assume that a clinician wants to retrieve all the patients with a diagnosis of brain tumor.

vID	date	patID	diag
1	20080201	3243	Brain Neoplasm
2	20080202	4722	Stomatitis
3	20080202	2973	Brain Tumor
4	20080204	9437	Skin Hemangiosarcoma
5	20080205	2437	Pineoblastoma

Figure 1: The table *Visit* recording patient visits

To search for such records, the clinician must (a) access the NCI thesaurus; (b) make a note of all the NCI synonym terms of brain tumor (in this case, there are 7 synonyms); and (c) use these terms in a query to the relational database to retrieve the relevant records (in this case, the records with vid's 1 and 3). Actually, this approach is familiar to clinicians since it is similar to the one used in PubMed/MeSH to retrieve medical articles (instead of patient records). However, the approach is clearly inefficient since it requires a lot of manual effort. The situation is even worse if the clinician also considers the hyponyms of brain tumor in order to retrieve patient records whose diagnosis refers to special cases of brain tumor (e.g., terms like *"Pineoblastoma"* or *"Thalamic Neoplasm"*). There are 233 such terms in NCI. It is practically impossible for the clinician to extract all this information manually from NCI in order to search for the appropriate records. Some level of automation is obviously required here.

Such automation would provide the clinician with a simple interface similar to popular search engines like Google. Then, the clinician would only perform the following steps: (a) indicate a medical term *QTerm*; and (b) specify whether, or not, the search should also consider the hyponyms of *QTerm* (synonyms of *QTerm* are considered, by default). In this paper, we show how to support such an ontology-based keyword search over a relational medical record database. Given the input term *QTerm*, our system *automatically* performs the following steps (i) it looks for *QTerm* in the ontology and, depending on step (b) above, it also collects the hyponyms of *QTerm*; and (ii) it uses the collected terms to retrieve the medical records. In the paper, we investigate several methods to provide the above functionality and we study the advantages and limitations of each method after extensive experimentation.

Method	Ontology Representation	Query	Maintenance
RDF Relational	`Thesaurus(src,rel,tgt).`	Recursive SQL for DAG traversal	Extract tuples from NCI Thesaurus CSV file or XML file
Native Relational	`Hyponym(src,tgt)`, `Synonym(src,tgt).`	Recursive SQL for DAG traversal	Extract tuples from NCI Thesaurus CSV file or XML file
Original XML	`ORG(vers,dat).` The original NCI Thesaurus XML file (Apelon schema) is stored in the dat column, in one row of the ORG table.	Use the SQL/XML function XMLTable to extract the hyponym links and use recursive SQL to traverse those links.	Just insert the new NCI Thesaurus XML file. No preprocessing required.
Hybrid XML Fragments	`HYB(vers,dat).` The dat column stores the concept XML fragments from the NCI Apelon XML file, one concept fragment per row.	Use the SQL/XML function XMLTable to extract the hyponym links and use recursive SQL to traverse those links.	Simple extraction of concept fragments from NCI Apelon XML file.
Hybrid XML tree	`HYBTree(vers,dat)`, `Synonym(src,tgt).` The synonyms has a flat structure and are stored in the Synonym relational table. The hierarchical hyponym relationships are materialized in a DAG and stored in a single XML file.	Use XPath for recursive traversals of subtrees and use recursive SQL to follow sub-tree refeerences.	Extract hierarchical hyponym relationships, materialize into a DAG in XML form. Extract synonyms into a table.

Table 1: Summary of the different methods for storing an ontology in a database.

2. Methods

Our method is summarized as follows. The ontology is first loaded into the database system possibly after some preprocessing. When the user wants to use the ontology to query the medical records, the only input necessary is the medical term *QTerm* and an indication of whether the hyponyms should also be considered while retrieving medical records. A database query on *both* the ontology and the medical records table is then executed and the results returned to the user.

There are several ways to support this type of processing depending on how the ontology is represented and stored in the database. Since XML and the Relational Model are currently the most popular models of representation, we consider both in this study. We assume that the EMRs are stored in a relational database, in a relation like the one in Figure 1. In practice, EMRs are often stored as relational records for efficient processing. Our method is equally applicable to EMRs stored in an XML format such as those using the HL7 CDA or CCR information models [3]. Unlike EMRs, ontologies are hierarchical in nature with the terms in the hierarchy often forming a *directed acyclic graph* (DAG). Therefore, ontologies can be represented using both XML and the relational model. We investigate five different methods of representing the ontology as summarized in Table 1. For ease of exposition, we use the NCI Thesaurus as our example ontology. Of course, our techniques are generic and applicable to any other ontology. Given term *QTerm*, for each method we present the automatically generated queries to retrieve the appropriate medical records. We do not require that the reader fully understands how these queries work. Instead, our aim is to illustrate how complex these queries are, no matter which method is considered, and thus prove the need for automation.

RDF Relational Method: In this method, a *single* relation, called *Thesaurus* is used to encode the whole ontology. Figure 2(a) shows some of the tuples in the *Thesaurus* relation storing the RDF-like representation of the NCI Thesaurus. Each triplet in the relation determines a relationship *rel* between a subject *src* and an object *tgt*. Of particular interest are the relationships *"hasSynonym"* and *"hasHyponym"*. The former is used to identify the synonyms of a terms. The latter is used to identify the hyponym terms of a term. Notice that each tuple records in the *tgt attribute* only one of the *immediate* hyponyms of the term stored in *src*. Therefore, a *recursive* query is necessary to identify and retrieve *all* the hyponyms, i.e., the hyponyms of the hyponyms, and then their hyponyms, and so on and so forth. Figure 2 shows queries $Q1$ and $Q2$, where the former query only retrieves patient records with a diagnosis that is synonymous to *QTerm* while the latter query also (recursively) considers the hyponyms.

src	rel	tgt
Brain Neoplasm	hasCode	C9344
Brain Neoplasm	hasHyponym	Intraventricular Brain Neoplasm
Brain Neoplasm	hasSynonym	Brain Tumor
Brain Neoplasm	hasHyponym	Supratentorial Neoplasm
Pineoblastoma	hasSynonym	PNET of Pineal Gland
Pineoblastoma	hasSynonym	Pineal PNET

(a) The *Thesaurus* RDF-like relation

```
SELECT DISTINCT V.* FROM Thesaurus T, Visit V WHERE src IN
  (SELECT src FROM Thesaurus WHERE tgt = 'QTerm' AND rel = 'hasSynonym')
  AND T.rel = 'hasSynonym' AND T.tgt = V.diag
```

(b) Query $Q1$ for synonyms

```
WITH Traversed (src) AS (
  (SELECT src FROM Thesaurus WHERE tgt = 'QTerm' AND rel='hasSynonym')
  UNION ALL
  (SELECT CH.tgt FROM Traversed PR, Thesaurus CH
   WHERE PR.src = CH.src AND CH.rel='hasHyponym'))
SELECT DISTINCT V.* FROM Thesaurus T, Visit V WHERE src IN
  (SELECT DISTINCT src FROM Traversed) AND T.rel = 'hasSynonym'
  AND T.tgt = V.diag
```

(c) Query $Q2$ for hyponyms

Figure 2: The *Thesaurus* RDF-like table and queries.

src	tgt
Brain Neoplasm	Intraventricular Brain Neoplasm
Brain Neoplasm	Supratentorial Neoplasm

The *Hyponym* relation

src	tgt
Pineoblastoma	PNET of Pineal Gland
Pineoblastoma	Pineal PNET

The *Synonym* relation

Figure 3: The *Synonym* and *Hyponym* relations

***Native* Relational Method:** Unlike the previous method where only a single relation is required, here multiple relations are created to encode the ontology. Intuitively, we generate a separate relation for *each* type of relationship between the terms of the ontology.

```
...
<conceptDef>
  <name>Pineoblastoma</name>
  <code>C9344</code>
  <id>9344</id>
  <namespace>NCI</namespace>
  <kind>Findings_and_Disorders_Kind</kind>
  <definingConcepts>
    <concept>Embryonal_Neoplasm_of_the_CNS</concept>
    <concept>Malignant_Pineal_Region_Neoplasm</concept>
    <concept>Pineal_Parenchymal_Cell_Neoplasm</concept>
  </definingConcepts>
  ...
  <properties>
    <property>
      <name>Preferred_Name</name> <value>Pineoblastoma</value>
    </property>
    <property>
      <name>Synonym</name> <value>PNET of Pineal Gland</value>
    </property>
    <property>
      <name>Synonym</name> <value>Pineal Gland PNET</value>
    </property>
    ...
  </properties>
</conceptDef>
...
<conceptDef>
  <name>Malignant_Pineal_Region_Neoplasm</name>
  <code>C3573</code>
...
</conceptDef>
...
```

```
SELECT DISTINCT V.* FROM Visit V WHERE V.diag IN
  (SELECT DISTINCT syn FROM XMLTABLE ('db2-fn:xmlcolumn("ORG.DAT")
   /terminology/conceptDef/properties[property/name/text()="Synonym" and
   property/value/text()="QTerm "]/property[name/text()="Synonym"]/value'
   COLUMNS syn CHAR(64) PATH'.') AS Temp)
```

(a) Query $Q5$ for synonyms

```
WITH Traversed (cls, src) AS (
  (SELECT R.cls, R.syn FROM XMLTABLE ('db2-fn:xmlcolumn("ORG.DAT")
   /terminology/conceptDef/properties[property/name/text()="Synonym" and
   property/value/text()="QTerm "]/property[name/text()="Synonym"]/value'
   COLUMNS cls CHAR(64) PATH'./parent::*/parent::*/parent::*/name',
           syn CHAR(64) PATH'.') AS R)
  UNION ALL
  (SELECT CH.cls, CH.syn FROM Traversed PR,
   XMLTABLE ('db2-fn:xmlcolumn("ORG.DAT")
   /terminology/conceptDef/definingConcepts/concept[./text()=$parent]/
   parent::*/parent::*/properties/property[name/text()="Synonym"]/value'
   PASSING PARENT.cls AS "parent"
   COLUMNS cls CHAR(64) PATH'./parent::*/parent::*/parent::*/name',
           syn CHAR(64) PATH'.') AS CH))
SELECT DISTINCT V.* FROM Visit AS V
WHERE V.diag IN (SELECT src FROM Traversed)
```

(b) Query $Q6$ for hyponyms

Figure 4: Fragment of the original XML format for the NCI Thesaurus and corresponding queries.

Figure 3 shows the distinct relations used to represent the synonym and hyponym relationships in the NCI Thesaurus. The corresponding queries $Q3$ and $Q4$ are quite similar to those for the RDF relational method and we omit them for brevity.

***Original* XML Ontology Method:** This method assumes that a domain expert provides an XML representation of the ontology. This is indeed the case for the NCI Thesaurus (our example ontology). This method requires the least effort since the only thing required is to download the ontology (for example, the NCI Thesaurus), and insert it *as is* in a database. Figure 4 shows a fragment of the NCI Thesaurus XML and Figure 4(b) shows corresponding queries $Q5$ and $Q6$ for retrieving patient records with a diagnosis that is a synonym or hyponym to *QTerm* respectively.

***Hybrid* XML Fragments Method:** Here, we decompose the original XML ontology tree into a number of XML fragments (sub-trees). One fragment is created for each term in the ontology. For example, for the NCI Thesaurus XML in Figure 4, each `conceptDef` element is extracted as a fragment and stored in a separate tuple (there are approximately 64000 `conceptDef` elements and hence that many tuples). The intuition is that in the majority of cases, only a part of the tree needs to be accessed to execute a query and therefore it is not necessary to always process the whole tree. By splitting the ontology tree, only the fragment trees corresponding to terms that are relevant to a query are accessed. Furthermore, by storing each fragment as a tuple in a relation we are taking advantage of relational database technology (like indexes) to reduce query processing times. The structure of the corresponding queries, $Q7$ and $Q8$, are quite similar to those of the original XML ontology method and are hence omitted for brevity.

***Hybrid* XML Tree Method:** Starting from the original XML ontology tree, we create (i) a *single* XML tree to encode the hyponym term relationship; and (ii) a synonym relation, like the one in Figure 3(b), to store the synonym term relationship. Intuitively, here we choose for each relationship the most *appropriate* model of representation. So, the hyponym relationship which is inherently hierarchical is represented as an XML tree, while the synonym relationship is represented as a relation of synonym term pairs. To retrieve patient records whose diagnosis is synonymous to *QTerm*, the same query for the native relational method is used. For hyponyms, $Q9$ in Figure 5 is used.

3. Results

We used IBM DB2 v9.1 [5], on a 4-CPU (3.2 GHz) server with 4GB of memory, to store both the patient records and the ontology (for all methods) since DB2 natively supports storing both relational and XML data. We used the NCI Thesaurus for our experiments along with one million synthetically generated patient visit records. Before we present our main results, we offer some useful NCI Thesaurus statistics. These statistics will help us to both interpret the results of the experiments and guide the design choices for this and other ontologies. The NCI Thesaurus is 112MB in size and contains approximately 64000 terms. Figure 6(a) shows the distribution of these terms per depth of the hypernym/hyponym hierarchy (with depth 0 denoting the root). Notice that the majority of nodes are between depths 4 and 9 and nodes with depths smaller than, or equal to, 4 are expected to have a large

```
<root_node>
  <Diseases_Disorders_and_Findings>
    <Diseases_and_Disorders>
      <Psychiatric_Disorder>
        <Anxiety_Disorder>
          <Post-Traumatic_Stress_Disorder>
          </Post-Traumatic_Stress_Disorder>
          <Neurosis>
            <Combat_Neurosis> </Combat_Neurosis>
          </Neurosis>
          <Organic_Anxiety_Disorder> </Organic_Anxiety_Disorder>
          <Separation_Anxiety_Disorder> </Separation_Anxiety_Disorder>
        </Anxiety_Disorder>
      ...
  </Diseases_Disorders_and_Findings>
  ...
</root_node>
```

(a) Fragment of the Hybrid XML Tree.

```
WITH Traversed (elem, attr) AS (
  (SELECT X.elem, X.attr FROM Synonym S,
     XMLTABLE ('db2-fn:xmlcolumn("XR.DAT")//.[fn:name(.)=$start]//.'
     PASSING BY REF S.src AS "start"
     COLUMNS elem CHAR(64) PATH'fn:name(.)',
     attr CHAR(64) PATH'if (fn:exists(.[@cp])) then ./@cp else "F"') AS X)
  WHERE S.tgt = QTerm
  UNION ALL
  (SELECT X.elem, X.attr FROM
     (SELECT * FROM TRAVERSED T WHERE T.attr='T') AS T1,
     XMLTABLE (' db2-fn:xmlcolumn("XR.DAT")
     //*[fn:name(.) = $n and @cp="MR"]//.'
     PASSING T1.elem AS "n"
     COLUMNS elem CHAR(64) PATH'fn:name(.)',
     attr CHAR(64) PATH'if (fn:exists(.[@cp])) then ./@cp else "F"') AS X))
SELECT DISTINCT V.* FROM Visit V WHERE V.diag IN
  (SELECT DISTINCT S.tgt FROM Synonym S, Traversed T)
  WHERE S.src = T.elem
```

(b) Query $Q9$ for hyponyms

Figure 5: *Hybrid* XML Tree and query

number of hyponyms. Conversely, nodes with depths larger than 9 are expected to have small numbers of hyponyms (we return to this point later). Figure 6(b) shows the distribution of terms with respect to their hyponym fanout (we only consider here *direct* hyponyms in the graph, not the recursive hyponyms). Approximately 4500 terms have only one direct hyponym, 2500 terms have two direct hyponyms, etc. For clarity, we stop reporting in the graph any fanouts larger than 20. However, there is one term for almost every value of fanout up to 150 (e.g., there exists a term in the graph with 150 direct hyponyms). Interestingly enough, there are also two *special* terms in the graph that have 2630 and 3025 terms as direct synonyms. These are the *"Non Current Chemotherapy Regimen or Association"* and the *"Retired Concepts"* terms. Finally, in Figure 6(c) we show the number of synonyms (fanout) per term. On average, each term has two synonyms. Specifically, around 22000 terms have no synonyms, while 18000 terms have 1 synonym and 11000 have two synonyms. Again for clarity, we do not report fanout values larger than 15 although there exist terms for almost each fanout value up to 78.

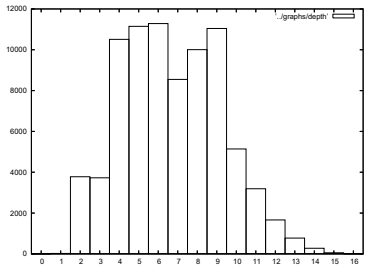
The experiments. For our first experiment, we used *QTerm = "Sebaceous Tumor"*. We consider this to be a *representative* term of the ontology since it has four synonyms (close to average number in the ontology), and it is at depth 8 (close to the *middle* of the ontological tree) with 53 hyponyms. Using *QTerm*, we executed *all* the queries, for *all* the methods, and in Figure 6(d) we show their running times. From the figure, it is clear that using the Original XML tree results in significant delays in query execution. Query $Q5$ (synonym query) requires 2 minutes to execute, while $Q6$ (hyponym) requires almost 4 1/2 hours. One the other hand, synonym queries for the other methods take as little as 30 msecs. For the hyponym queries, they require as little as 2 secs (for the relational only methods) and at most 30 secs (for the hybrid tree method). This vast difference in query execution between hours/minutes and secs/msecs clearly proves our claim that the initial XML ontology is not designed for efficient query processing and some form of post-processing is necessary to convert it to a more query-friendly form. The above also illustrate the value of our solutions in this and more generic settings.

In our second experiment, we study the effect of the number of hyponyms on query evaluation time. We consider three terms, namely, *"Non-Neoplastic Disorder"*, *"Colon Neoplasm"* and *"Plasmablastic Lymphoma"*, which are in depths 3, 8 and 13, respectively. Remember that the smaller the depth, the higher in the tree the term is, and therefore the more hyponyms the term is expected to have. Indeed, the three terms have 1901, 64 and 8 hyponyms, respectively. Figure 6(e) shows that for both the relational only methods, the evaluation time is proportional to the number of hyponyms, which verifies our intuition. We do not show the XML methods, since their running times are 2-3 orders of magnitude larger than the relational ones.

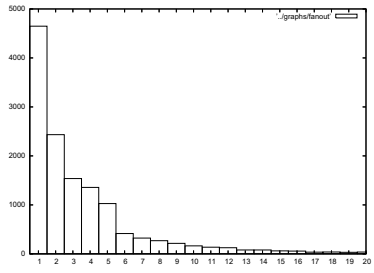
For our last experiment, we study the effect of the number of synonyms on query evaluation time. We consider three terms, namely, *"Stomatitis"*, *"Skin Angiosarcoma"* and *"Prostate Cancer Stage III"*, which have 1, 5 and 10 synonyms, respectively. Figure 6(f) shows that evaluation times are also proportional to the number of synonyms, for the RDF-like, Native and Hybrid fragment methods. Concerning the Original XML method, the situation is similar although the evaluation time is around three orders of magnitude bigger (the three methods in Figure 6(f) require fractions of a second, while the evaluation times of the Original XML method is between 1 and 1 1/2 minute).

4. Discussion

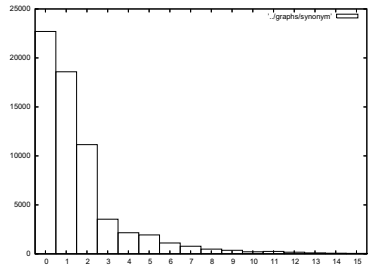
The previous sections present a system with a simple (keyword-based) search engine-like query interface to a clinician (often a computer non-expert), where a simple term is automatically expanded to also consider its synonyms (and possibly, its hyponyms). Our investigation has shown that considerable tangible gains are possible if we store and process an ontology in



(a) Term distribution per tree depth



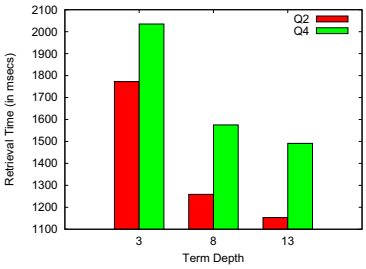
(b) Term distribution per fanout



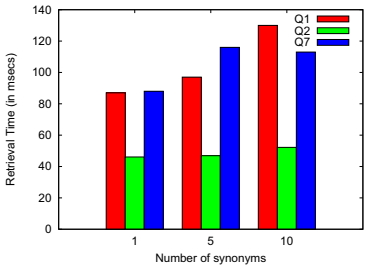
(c) Term distribution per synonyms

	Method - Scope	Time (ms)
Q1	RDF Relational - Syn.	38.6
Q2	RDF Relational - Hyp.	1376.8
Q3	Native Relational - Syn.	32.9
Q4	Native Relational - Hyp.	1543
Q5	Original XML - Syn.	116,422.7
Q6	Original XML - Hyp.	15,778,948
Q7	Hybrid XML Frag. - Syn.	46.4
Q8	Hybrid XML Frag. - Hyp.	2577.7
Q9	Hybrid XML Tree - Hyp.	31,597.8

(d) Evaluation of representative *QTerm*



(e) Evaluation at different depths



(f) Evaluation with varying synonyms

Figure 6: Ontology statistics

a way that is more appropriate for querying. These gains however must be weighted with the associated cost of maintaining the ontology up-to-date. Specifically, for each of the proposed methods, one must consider what are the effects of *updating* the ontology tree. One would expect that the method of using the Original XML Ontology tree would require the least *effort* since the only thing required is to replace in the database the previous version of the tree with the new (updated) one. Surprisingly, given the size of the ontology tree, this simple operation can take in the order of a few hours to complete during which time the system becomes unavailable for querying. The crucial observation here is that no matter how big, or small, an update to the tree is, the whole tree needs to be replaced. In the worst case, even correcting a simple typo in a term of the ontology requires to take the whole system offline for a few hours. Obviously, this is not a very satisfactory solution.

Unlike the previous method, the hybrid fragment and relational-only methods allow faster loading times. In these methods, the original tree is decomposed to a number of tuples and smaller subtrees. Therefore, one can isolate the parts of the tree that have been updated and only replace those in each method.

5. Related work

There has been a lot of interest on developing systems to support keyword search over a relational database. In systems like BANKS [2], DBXplorer [1] and Discover [4], the user provides as input a set of keywords and the system returns as answer a set of *tuples trees* (joinable tuples from multiple relations). However, a requirement in these systems is that *all* the input keywords are present in *each* returned tuple tree. In our work, the semantics is quite different since we return a tuple if it mentions at least one of the keywords. Another key distinction between the works is that these systems do not exploit any ontologies that are relevant to the input keywords. In contrast, our work uses such ontologies to automatically expand the initial set of keywords (and retrieve more relevant results) without requiring any additional user effort.

References

[1] S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *ICDE*, pages 5–16, 2002.

[2] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *ICDE*, pages 431–440, 2002.

[3] J. M. Ferranti, R. C. Musser, K. Kawamoto, and W. E. Hammond. The clinical document architecture and the continuity of care record: A critical analysis. *JAMIA*, 13(3):245–252, June 2006.

[4] V. Hristidis and Y. Papakonstantinou. Discover: keyword search in relational databases. In *VLDB*, pages 670–681, 2002.

[5] IBM DB2 Universal Database pureXML. http://www-306.ibm.com/software/data/db2/xml/.