Keyword Search Across Databases and Documents*

Carlos Garcia-Alvarado
University of Houston
Dept. of Computer Science
Houston, TX, USA

Carlos Ordonez
University of Houston
Dept. of Computer Science
Houston, TX, USA

ABSTRACT

Given the continuous growth of databases and the abundance of diverse files in modern IT environments, there is a pressing need to integrate keyword search on heterogeneous information sources. A particular case in which such integration is needed occurs when a collection of documents (e.g. word processing documents, spreadsheets, text files and so on) is derived directly from a central database, and both repositories are independently updated. Finding hidden relationships between documents and databases is difficult, given the loose connection between them. This problem is especially complicated when database integration techniques must be extended to handle semi-structured data (i.e. documents). Our research focuses on exploiting a relational database system for integrating and exploring complex interrelationships between a database and a collection of potentially related documents. We focus on the discovery and ranking of keyword links (relationships) at different granularity levels between a database schema and a collection of documents. We adapt, extend, and combine information retrieval techniques into the DBMS. As such, we provide algorithms for efficient exploration of discovered relationships among a collection of documents and a DBMS. We experimentally show that our system can discover, query and rank complex relationships discovered between a database and surrounding documents.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data Mining*

General Terms

Algorithms, Experimentation

Keywords

Keyword Search, Database, Documents

*This research work was partially supported by NSF grants CCF 0937562 and IIS 0914861.

http://doi.acm.org/10.1145/1868366.1868368

1. INTRODUCTION

Keyword search has become a ubiquitous concept for end-users since the emergence of web search engines such as Yahoo! or Google. As a result, traditional query languages for structured sources such as SQL appear to be a complicated option for inexperienced users whose desire is to be able to query without knowing the structure of the data. Keyword search in databases has been widely explored in recent papers [7, 3, 1, 9, 16, 4]. However, there are multiple unstructured collections whose origin is a structured source.

A typical example of a collection of documents created from a structured source is found in the registrar's office at a university. It has a central database with information about current students in the university, including student ID, name, major, grades, and financial information, among others. Aside from this database, each department and professor has documents about these students, such as medical information, homeworks, reports, theses and papers, and many more types of documents. These unstructured sources are all modified and created independently, without the database's support. Nevertheless, the links between both sources are no longer obvious, and there is an increasing need to perform searches that will allow us to relate and rank these data sources again. Hence, a way to relate these two areas represents an immediate need for the organization.

Despite the fact that the scenario is clear, a number of challenges arise in solving such a complex problem of relating a database with a collection of documents. Some of these problems involve discovering links between the document's data and the database in the DBMS. An initial way of finding these links is by performing keyword matching between both sources. As a result, our objective in this paper is to achieve structured and unstructured data linkage using elements residing in the DBMS, and to allow the user to explore and perform searches in both data sources.

The information stored in a database system contains structure and hierarchy. Hence, the data stored in a database is already associated with an explicit meaning, as compared to the data spread throughout unstructured sources. In addition, most data stored in the DBMS undergoes a validation process in order to guarantee the integrity constraints[13]. On the other hand, the semistructured and unstructured sources do not undergo this validation process, and they are modified independently of one another. Without loss of generality, for the purpose of this paper, all semistructured and unstructured documents are going to be considered indistinguishable. These unstructured documents are created outside of the "rigorous" structure and enforcement of cor-

rectness given by the referential integrity of the database (controlled tabular environment) with information explaining the content of the database. This lack of validation has made the task of matching difficult. Despite this, the data that reside outside the database system have become valuable, mostly due to the ability to describe these structured data in further detail. We claim that it is possible to explore, search, and in some cases rank, how these portions of the database are related to documents, and at the same time provide some typical Information Retrieval (IR) capabilities to the user for querying medium-sized collections of interrelated documents with an acceptable performance.

The organization of this paper is as follows: In Section 2, we present the common notation and basic definitions for the rest of the paper. In Section 3, we describe our novel proposal for linking structured and unstructured sources, and our proposed data layout and implementation for managing ranked relationships and searching for results. Section 4 presents an analysis on the feasibility and performance of the implementation of our approach, and discusses the experimental results. In Section 5, we describe some previous research in the areas of keyword search and schema matching. Finally, in Section 6, we present our final remarks.

2. DEFINITIONS

We focus on keyword search between a central database and a collection of documents. As a result, the definitions include elements from both domains. In the structured domain, let a database D be a set of m tables t, in which each table contains a set of columns c and $|t_i|$ rows. In addition, let e be an element stored in D in a level of granularity γ. We assume that every element in the DBMS can be mapped directly to a keyword k with an average size of $\bar{k}$. Moreover, let the granularity level γ address a table, column or record in D. As a result, an element in the database e with a granularity level equal to a row will require an address in the database represented by a set of primary keys and column references, or a reference to a column in the column level or a reference to a table in the table level. In the unstructured domain, let C be a corpus of $\bigcup_{i=1}^{N} d_i$ documents residing around D. In addition, let $Appx_k$ be a set of all the approximate keywords of k given that the edit distance of keyword k' from keyword k is less than or equal to $MAX(1, \beta * |k|)$, where β is a real number between 0 and 1. Intuitively, we can safely assume β to be about 10% to obtain good approximation results for mistyped keywords. Moreover, let δ be the edit distance and let $EditDistance(k, k')$ be a boolean function that returns true if k' is an approximate keyword of k. Following the basic definitions, let us focus on the linkage of both sources. In order to do so, we propose some definitions for obtaining such "links".

***Definition** 1.* Let S be the superset of all the possible links between D and C.

The universe of all possible links that exist between both sources is contained in S. As such, S contains links that can be inferred from analyzing both collections, which are the focus of this paper, and some others that are the result of links given by some previous knowledge of both sources (e.g. a user-defined object mapping). By extending the definition of e, let $e =< k, \gamma >$. A link between a database element and a document keyword is as follows:

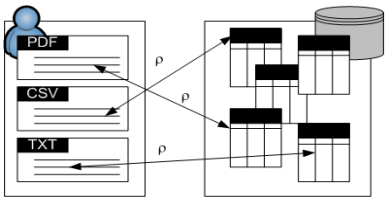

Figure 1: Relationships ρ_γ.

***Definition** 2.* Let ρ be a relationship representing a link between a document keyword and an e based on the finding of approximate keywords of a given element in D.
$\rho = (e, d_i) \in S | Appx_k \neq \emptyset,\ x \rightarrow k.$

These newly inferred "links" now are identified as relationships between both sources. As a result, we will focus on finding all the e that are related to all the keywords in the documents (see Figure 1). Moreover, let L be a set of all unique keywords existing in D without considering γ and let E be the set of all the elements in the database considering γ. The construction of E is the result of creating a set of unique keywords L and a dictionary that references each k to a granularity level.

Notice that the ρ_γ (relationships in each granularity level) are independent sets. As a result, a relationship existing between a keyword of a granularity level in the database may not exist in another granularity level. Despite this, by the transitive properties that exist between the granularity elements in the DBMS (e.g. a table contains attributes and records), we can "assume" new relationships by performing aggregations. However, we do not consider these types of inferences to be direct links between elements in the DBMS and documents. For example, assume a small database with a single $T_1(a_1,a_2)$ and a corpus of a single document $d_1 = \{a_1\}$. The number of relations inferred from both sources will be $\rho_1 = (a_1, d_1)$. With further aggregations, we could assume that there is a new relation $\rho_2 = (T_1, d_1)$; however, this type of derived relationships will not be considered relevant for this work.

Once these relationships have been found, we focus on querying these links. In this paper, we propose two methods for querying the set of ρ. A initial method is performed by returning all the ρ sorted by the importance of the element based on γ, δ to the searched keyword and finally by the importance of the link between the keyword and the document. A second method also relies on the importance of the element in the database and δ of the keyword to the searched element. However, we use a more complex weighting scheme, in which the importance of a document for a given keyword will be defined as the average of the edit distance multiplied by the frequency of approximate keywords in each document.

3. KEYWORD MATCHING

The process of extracting and querying all the relationships is the result of a preprocessing phase, which includes the inference of the relationships, and a querying phase of the final data structures. The preprocessing phase, or relationship construction, obtains all the ρ_γ relationships and stores them in an efficient data structure for querying. The second phase presents an algorithm for querying and ranking

```
Input: D, C, β
01 : L ← ∅, E ← ∅,R ← ∅
02 : foreach k in D
03 :     if( k ∉ L )
04 :         L ← L ∪ {< k >}
05 : foreach k in L
06 :     t_k ← 0,dt_k ← 0
07 :     foreach d_i in C
08 :         Appx_k ← ABM(k, d_i, β)
09 :         if(Appx_k ≠ ∅)
10 :             R ← R ∪ {< k, d_i, |Appx|, |Appx|δ̄ >}
11 :             t_k ←t_k + |Appx|
12 :     foreach p in R_k
13 :         w ← w/t_k|w ∈ p
14 :     if( t_k ≠ 0 )
15 :         foreach k' in D
16 :             if( k = k' )
17 :                 E ← E ∪ {< k, γ >}
18 :     else
19 :         L ← L − k
20 : return R, E
```

Figure 2: Relationship Construction.

the final results.

3.1 Matching Algorithm

The relationship inference can be seen as seen as a simple but costly operation defined by querying every element of the database and every document in the collection. However, such an approach is extremely inefficient, and will result in a large space and time complexity. As a result, the purpose of our new algorithm is to reduce the number of elements to be searched and maximize the number of relationships that can be found between the two data sources.

The maximization of the number of keywords that can be found in a document was obtained by performing an approximate string matching instead of assuming a perfect match scenario. The Approximate Boyer-Moore algorithm (**ABM**) described in [15] was adapted to search for approximate keywords in the collection of documents in the DBMS. In addition, the number of mismatches allowed by the algorithm to be considered as an approximate match is controlled by a β parameter, which guarantees that the number of mistakes is proportional to the size of the keyword instead of fixing the number of mismatches.

In Figure 2, we present an efficient approach for discovering all the relationships. The input for the algorithm is a Database D, a Corpus C and parameter β for the ABM algorithm. In line 01, the output sets are initialized to the empty set. R contains all the matches between a keyword and a document p existing in C. L contains all the different keywords existing between the elements. E contains all the elements of the database and is also initialized to the empty set. Lines 02-04 obtain the set of all the possible elements to look for (in their respective granularity levels) and obtains the set of unique keywords to search. Lines 05 to 19 iterate through all the keywords in L to search, and find all approximate keywords that exist in the corpus of documents. In line 11, the t_k temporary variable stores the cumulate value of the number of approximate matches that link a document to a keyword. As such, in line 13 a weight is assigned to each document for every relationship. Between lines 15-17 each existing keyword is matched against any existing element in the database. This step is performed at this stage to guarantee that E occupies the least space possible. Line 19 removes the keywords that could not be found

```
Input: Q, R, E
01 : Result ← ∅, Presult ← ∅
02 : foreach p in R
03 :     foreach k in Q
04 :         if(EditDistance(k', k)|k' ∈ p)
05 :             R' ← R' ∪ {< p, δ >}
06 : foreach< p, δ > in R'
07 :     foreach e in E
08 :         if(k ∈ p ∧ k ∈ e)
09 :             Result ← Result∪
09 :                 {< ρ, δ, w_{k,i}δ̄_{k,i}_>}
10 : Rank Result by γ, δ and w_{k,i}δ̄_{k,i}
11 : return Result
```

Figure 3: Relationship Querying.

in the collection of documents. Finally, line 20 returns the only important data structures that are required: a set of all the elements in the database, and the existing relationships and their weights.

3.2 Keyword Search

Querying the discovered relationships requires searching only over the newly extracted relationships. As such, Figure 3 presents an efficient way to traverse the R and E data structures. The input of this algorithm is a set of keywords $Q = \{k_1, k_2, \dots\}$. In line 01, the Result set and a partial result container are initialized to the empty set. Lines 02 to 05 represent a join for extracting the keyword-document pairs p in R that have keywords in a valid approximate distance to a given query Q. Notice that this step can be performed more efficiently if R is reduced by using L as an additional data structure for indexing.

All the valid document-pairs p are stored in a partial set. Lines 06-09 match the relationships with the elements residing in the DBMS. Notice that all of these joining steps are presented in polynomial time. However, this time can be reduced to close to linear by using hashes or indexes. Line 10 sorts the *Result* set by the granularity level, the edit distance of the keyword and the weight of each relationship. The last step returns the sorted set.

3.3 Complexity Analysis

The processing algorithm can be divided into two steps that need to be performed sequentially. The first step is the storage of the unique set of keywords L from the database. The second step is the extraction of all the ρ relationships. A breakdown of the complexity of all the steps of the algorithm in Figure 2 is described in Table 1. The first step is the extraction of the unique step of keywords from the database, which is performed in the time it takes to scan all the elements in the database. The second step is the computation of R, which is dependent on the size of $|L|$, the average size of the keyword to search $\bar{k}$ and the average size of a document in the collection. Notice that this step represents the bottleneck of the time complexity of the algorithm. The extraction of all the valid matches between elements in D and E is dependent on the number of keywords with matches in the documents $\hat{L}$ given by ABM. In the worst case, $|\hat{L}|$ will be equal to $|L|$. Thus the total complexity is $\mathcal{O}(m|t_i| + |L|(ABMp + |C|ABMs) + |\hat{L}|m|t_i|)$. Despite the number of variables in the algorithm, the most important variable is the set of unique keywords L.

In Table 2, the breakdown of the search algorithm is described. The first step consists of obtaining the valid rela-

Table 1: Complexity of the Construction Algorithm.

Step	Complexity
L	$\mathcal{O}(m\|t_i\|)$
ABM-preprocess	$\mathcal{O}(\bar{k} + \beta\bar{k})$
ABM-search	$O(\|\bar{d}_i\|\beta\bar{k}(1/(\bar{k} - \beta\bar{k})) + \beta\bar{k})$
R	$\mathcal{O}(\|L\|(ABMp + \|C\|ABMs))$
E	$\mathcal{O}(\|\hat{L}\|m\|t_i\|)$

Table 2: Complexity of the Querying Algorithm.

Step	Complexity
R'	$\mathcal{O}(\|R\|\|Q\|\bar{k}^2)$
$Result$	$\mathcal{O}(\|R'\|\|E\|)$
Sort	$\mathcal{O}(\|Result\| \log \|Result\|)$

tionships that are within a valid edit distance from any of the given queries in the keyword. The search complexity can be optimized by reducing the size of R with a preselection and then performing the loop. The second phase of the search requires matching the elements in the database with the keyword-document pairs. The last step requires a sort based on the edit distance, the granularity level and the importance of d_i. The final complexity of the search is $\mathcal{O}(|R||Q|\bar{k}^2 + |R'||E| + |Result| \log |Result|)$, in which early selection and hashes can improve these "joins".

3.4 DBMS Implementation

When implementing both algorithms in the DBMS, we need an efficient storage and retrieval structure. A database schema for storing the relationships is shown in Figure 4. The existing elements of the DBMS (stored in E) are contained in the `table_level`, `column_level` and `row_level` tables. The layout for storing each relationship type has different requirements in order to store their location in the database. As such, a different vertical format is used for each type. For example, primary keys' values are different among tables. The `element-keyword` table maps all the existing elements e into a keyword. The L set is stored in the `keyword` table. The document collection is stored in the `collection` table. The resulting keyword-document relations R are stored in the `relationship` table. A final weight is assigned to each relationship based on the number of approximate matches found in a specific keyword and the collection. In order to efficiently extract the type of relationships, indexes must be created in the PK/FK relations. Additional indexes on keyword id *kid* must be included to speed up the join conditions in the retrieval.

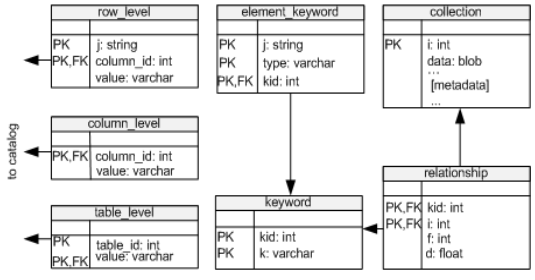

Figure 4: Relationship Data Layout.

The relationship discovery phase requires a process for querying the catalog and finding all the elements that exist in the DBMS. A Stored Procedure that iterates through all the tables extracts the unique set of keywords. An additional process is required to validate whether or not this keyword is a valid element (e.g. numbers are ignored). This step is necessary when traditional search engines discard stopwords or nonrepresentative words in the text. For further reference, see [5]. The relationship discovery process is then performed in two queries in the DBMS. The first query takes the `keyword` table and analyzes each document through a Table-Valued Function (TVF). This TVF performs as many of the computations in main memory as possible, in order to speed up the execution. Hash tables are used as backbone data structures for computing the summarizations required in this step. The following step removes all the keywords without any occurrence. A final query in the preprocessing step is performed to populate the `element_keyword` table and each element table. This query compares the elements in the DBMS to the list with possible matches.

The retrieval of the related relationships is performed with an initial search in the `keyword` table to obtain the valid keywords using a user-defined function for the edit distance. The last query obtains the relationships from a hash join based on the keyword id.

4. EXPERIMENTS

We conducted our experiments on a computer with an Intel Core Duo E8400 processor at 2.53 GHz. The system hardware configuration also has 8 GB of RAM and a hard drive with 1.5 TB of space. The software configuration included a running instance of Microsoft SQL Server. Our application was developed using C# for dynamically generated SQL as a thin front-end for running queries. The relationship construction algorithm and the relationship query algorithm were developed using User-Defined Functions (UDFs) and Stored Procedures (SP).

Our motivation for developing this novel approach was to find a reduced set of relationships in complex database schemes which can be stored, used, and efficiently managed in a DBMS, instead of using the original document collection. As a result, we decided to test our approach with two data sets in different domains. The first set is a real scientific database containing water pollution data from wells in the state of Texas (TWWD), and a collection of public documents downloaded from the Texas Commission on Environmental Quality (see Table 3). In addition, a second data set of documents was created from the the ACM Digital Library and a medium-sized database was built from DBLP (see Table 4). For practical purposes, we ignored keywords smaller than three characters, stopwords, symbols, and numbers.

4.1 Relationship Discovery

The initial set of experiments focuses on scalability and performance of extracting the relationships between a corpus and a database. In Table 5, the number of relationships between C and D are presented. Table 6 shows an analysis of the time performance of the relationship construction algorithm. The first set of experiments shows how the element-keyword relation does not vary significantly once the size of the collection increases. This is due to the fact that the increase is only produced in the relationship table.

Table 3: Texas Water Wells Data Set C and D.

Description	Value	Min	Max
Corpus			
Documents	1000	-	-
Avg. keywords per d_i	217	1	250
Database			
No. of Tables	32	-	-
No. of Columns	214	-	-
No. of Elements	36892343	-	-
Avg. number of rows per t	111488	0	706223
No. of different k	278408	-	-

Table 4: ACM DL Data Set C and D.

Description	Value	Min	Max
Corpus			
Documents	1000	-	-
Avg. keywords per d_i	206	53	236
Database			
No. of Tables	3	-	-
No. of Columns	9	-	-
No. of Elements	462778	-	-
Avg. number of rows per t	469902	74498	1216779
No. of different k	190233	-	-

As a result, we observe that E increases in smaller intervals. Depending on how "descriptive" the schema is, it may happen that a few or none of the relationships will be related to an element in the tablename or columnname granularity levels. For example, in the ACM DL collection, all of the relationships were found in the row level.

In the second set of experiments, we observe that the bottleneck of the algorithm for the TWWD collection is found during the storage and extraction of the elements in the database. Similar to the number of elements in D, the performance times for extracting these elements does not vary significantly. However, in the ACM DL collection, the bottleneck occurs in the relationship computation. The rationale behind this is that the number of elements in the database was much smaller in the ACM DL than in the TWWD collection. In addition, we observed that 99% of the time for building the E set was spent on obtaining the elements contained in the lowest level of granularity. We noticed that the algorithm has an acceptable performance when obtaining all of the relationships. Despite this, the performance is highly tied to the size of L, as we observed in the complexity analysis. It is worth mentioning that the preprocessing step is only executed once for every set of documents, and is easily manageable for adding more documents incrementally.

4.2 Relationship Querying

In the second section of the experiments, we focus on querying the extracted relationships in the preprocessing step. This was achieved by selecting a set of 20 random queries from the keyword table for each subset of documents. Table 7 shows the average, the maximum, the minimum, and the standard deviation performance times required by each set of queries in every collection and subset of documents. In these sets of experiments, we noticed that the query times were quite efficient and similar in all cases for both ranking techniques (we only show the results for the first ranking technique due to space limitations). In particular, we observed that the querying times are approximately steady between the set of documents in the same collection. However, as expected, the times increase when the number of elements found is larger. The rationale behind this behavior is that the first query in the data set has to verify the keywords that are in a valid edit distance. As a result, a full scan in the `keyword` table is required. Once this step has been performed, a join between the relationship table and the `element-keyword` table on the indexed *kid* columns is performed. As a result, the time for the join (hash joined) is quite efficient, and the bottleneck of the query search becomes the early reduction of the keyword table which requires a full table scan.

Table 5: Relationship Construction.

C	e in t	e in c	e in r	R
TWWD				
100	7	30	2406799	28768
250	9	41	2425013	72408
500	10	47	2437147	143820
1000	11	53	2451387	282364
ACM DL				
100	0	0	8336	121908
250	0	0	8336	147071
500	0	0	8741	293583
1000	0	0	9305	595829

Table 6: Construction Performance (in sec).

C	E	L	R	Total
TWWD				
100	240	33	24	297
250	341	48	46	435
500	359	48	85	492
1000	386	48	183	617
ACM DL				
100	119	6	119	244
250	122	5	132	259
500	122	5	179	306
1000	138	6	351	495

5. RELATED WORK

Searching keywords between a database and a collection of documents has been approached slightly by keyword search and schema matching. Schema matching is an important research area that has been explored for quite a long time [14]. Early attempts were made in [10], in which the authors try to match both schemes using a set of given rules. In [2], the authors describe a system called MOMIS, which uses an Object Data Model (ODM) that relies on a semantic approach to match the structured and unstructured sources. In [8], an architecture for creating a collaborative centralized infrastructure for sharing scientific data is proposed. The architecture uses a search schema that allows users to locate data sets by exploiting metadata information. In contrast to schema matching, keyword search in databases has centered on finding keywords in all of the elements in the database. Different approaches have been developed using tuple trees and joins

Table 7: Relationship Querying (in ms).

C	Mean	StdDev	Max	Min
TWWD				
100	327	16	343	312
250	437	189	656	328
500	312	12	328	296
1000	320	20	343	296
ACM DL				
100	40	14	62	31
250	34	7	46	31
500	40	8	46	31
1000	37	8	46	31

[7, 3, 1, 9, 16, 4]. Finding links among these two sources is not directly schema matching or keyword search exclusively in the DBMS. However, ideas from both topics have been improved for achieving this task. To the best of our knowledge, EROCS is one of the few attempts at obtaining database-document linkage [4]. EROCS focuses on linking transactions and then performing data mining operations over the newly discovered links. The linkage is performed using templates for identifying entities. Finally, in [12, 6], the authors suggest a method for relating schema-keywords with terms in the documents, and extracting macro and micro relationships. The matching between both sources was done using an ODM, as presented in [2], and the unstructured sources were managed using XML. We integrated the schema matching and keyword search ideas, and extended the papers by [12, 4, 6] to present a more complete analysis on the existing types of links between elements in the DBMS and documents and we present a novel idea on how to rank these relationships within the DBMS. In addition, we do not require a given ODM for discovering these links, and we perform the data linkage without any entity templates. Moreover, clustering techniques can be used to obtain better linking elements in the DBMS [11].

6. CONCLUSIONS

In this paper, we present a process for discovering "links" between the elements of a central database and a collection of documents. The links are defined as relationships in the tablename, columnname and record level of granularity (ρ_γ). In addition, an algorithm for discovering such relations was presented. We found that that the bottleneck of the algorithm is in searching for the elements in the database when a set of keywords is given. In addition, the search of approximate keywords appears to be a target for future optimizations. Moreover, an algorithm for searching in the newly discovered relationships was presented. This algorithm performed in an efficient manner, due to the fact that the search is only performed in this smaller data structure containing the set of valid relationships. Also, the proposed data layout, indexing and querying strategies, optimized the search.

Future work in this research includes applying top-k techniques for faster retrieval of the relationships. In addition, we noticed that this algorithm can be computed in parallel, which can speed up the execution of both tasks (preprocessing and retrieval). The optimization for approximate string matching with multiple keywords appears as an interesting area of research. Also, the extraction of relationships resulting from the PK/FK relations in the database schema may return additional interesting results. Finally, an analysis on the derivate result obtained from the transitive properties of the database schema (e.g. row belongs to a table) can give results complementing the ones explored in this paper.

7. 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] S. Bergamaschi, S. Castano, and M. Vincini. Semantic integration of semistructured and structured data sources. *ACM Sigmod Record*, 28(1):54–59, 1999.

[3] 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.

[4] M. Bhide, V. Chakravarthy, A. Gupta, H. Gupta, M. Mohania, K. Puniyani, P. Roy, S. Roy, and V. Sengar. Enhanced Business Intelligence using EROCS. In *ICDE*, pages 1616–1619, 2008.

[5] C. Garcia-Alvarado and C. Ordonez. Information retrieval from digital libraries in SQL. In *ACM WIDM Workshop*, 2008.

[6] C. Garcia-Alvarado, C. Ordonez, and Z. Chen. DBDOC: Querying and Browsing Databases and Interrelated Documents. In *ACM KEYS SIGMOD Workshop*, 2009.

[7] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword Search in Relational Databases. In *VLDB*, pages 670–681, 2002.

[8] A.R. Jaiswal, C.L. Giles, P. Mitra, and J.Z. Wang. An architecture for creating collaborative semantically capable scientific data sharing infrastructures. In *ACM WIDM Workshop*, pages 75–82, 2006.

[9] F. Liu, C. Yu, W. Meng, and A. Chowdhury. Effective keyword search in relational databases. In *SIGMOD Conference*, pages 563–574, 2006.

[10] T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. In *VLDB Conference*, pages 122–133, 1998.

[11] C. Ordonez. Models for association rules based on clustering and correlation. *Intelligent Data Analysis*, 13(2):337–358, 2009.

[12] C. Ordonez, Z. Chen, and J. García-García. Metadata management for federated databases. In *ACM CIMS Workshop*, pages 31–38, 2007.

[13] C. Ordonez and J. García-García. Referential integrity quality metrics. *Decision Support Systems Journal*, 44(2):495–508, 2008.

[14] E. Rahm and P.A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*,10(4):334–350, 2001.

[15] L. Salmela, J. Tarhio, and P. Kalsi. Approximate Boyer-Moore String Matching for Small Alphabets. *Algorithmica*, 58(3):591–609, 2010.

[16] M. Sayyadian, H. LeKhac, A.H. Doan, and L. Gravano. Efficient keyword search across heterogeneous relational databases. In *ICDE*, pages 346–355, 2007.