ENHANCING KEYWORD SEARCH OVER RELATIONAL DATABASES USING ONTOLOGIES

Ahmed Elsayed[1], Ahmed Sharaf Eldin[1], Doaa S. El Zanfaly[1, 2]

[1]Information Systems Department, Faculty of Computers and Information,
Helwan University, Cairo, Egypt
eng_ahmedyakoup@yahoo.com, profase2000@yahoo.com,
doasad71@yahoo.com
[2]Informatics and Computer Scinence, British University in Egypt, Cairo, Egypt
doaa.elzanfaly@bue.edu.eg

ABSTRACT

Keyword Search Over Relational Databases (KSORDB) provides an easy way for casual users to access relational databases using a set of keywords. Although much research has been done and several prototypes have been developed recently, most of this research implements exact (also called syntactic or keyword) match. So, if there is a vocabulary mismatch, the user cannot get an answer although the database may contain relevant data. In this paper we propose a system that overcomes this issue. Our system extends existing schema-free KSORDB systems with semantic match features. So, if there were no or very few answers, our system exploits domain ontology to progressively return related terms that can be used to retrieve more relevant answers to user.

1. INTRODUCTION

A significant amount of plain text and structured data has been stored side by side in relational databases for decades. In order to query this data, users have to know the database schema and then use Structure Query Language (SQL) to issue a precise, unambiguous and well-formed query. To relieve users from doing this, Keyword Search over Relational Database (KSORDB) enables casual users to query this data using a set of keywords called keyword query. Existing KSORDB approaches can be categorized into two main categories: Schema-based approaches [1-3] and Schema-free approach [4-7]. In order to process a keyword query, the schema-based approach uses the schema graph to generate Candidate Network (CNs) and then evaluate these networks using SQL queries. While in the second approach, schema-free, the database is modeled as a data graph where nodes are tuples and edges are foreign - primary key relationships. The graph is then traversed, at the query time, to answer keyword queries.

The main difficulties with the schema-based KSORDB systems are in generating optimal SQL queries from a huge number of CNs. Moreover, the generated queries usually contain many join

Sundarapandian et al. (Eds) : ACITY, AIAA, CNSA, DPPR, NeCoM, WeST, DMS, P2PTM, VLSI - 2013
pp. 147–154, 2013. © CS & IT-CSCP 2013 DOI : 10.5121/csit.2013.3415

operations which are characterized by complex processing and time consumption features. Schema-free KSORDB systems have three main drawbacks: First, they are not efficient for large size databases as they consume the memory in materializing and processing the data graph. Second, the data graph should be updated each time the underlying database is updated; therefore, this model is not appropriate for databases that change frequently. Third, traversing a huge data graph to find minimum Steiner trees is proved to be an NP-hard problem [8]. Furthermore, both KSORDB approaches implement exact keyword matching without exploiting the semantic relationships, such as such as meronym, hyponymy, and antonym, between keywords to find the correlated data.

In Information Retrieval (IR) community, Ontology is used to support semantic matching. "An ontology is an explicit specification of a conceptualization" [9]. Therefore, ontology is used to relate concepts in a specific domain. Keyword search over relational database techniques can exploit ontology to support semantic matching.

In this paper a KSORDB system based on the schema-free approach is proposed. The system does not need to maintain a whole data graph in memory as it pre-computes and materializes the connectivity information offline in a data structure called reachability index. In addition, the proposed system also supports the semantic matching by exploiting the structure of domain ontology.

The rest of this paper is organized as follows: section 2 discusses related work. Whereas, section 3 describes the proposed system architecture. A review of Ontology-based semantic similarity methods is given in section 4. Finally, section 5 discusses the performance and enhancement expectation of the proposed system

2. RELATED WORK

Extensive work has been done in the area of keyword search over relational database. This work can be categorized into two main categories according to the data models they adopted: Schema-based approaches [1-3] and Schema-free approaches [4-6]. The former use schema graph information to compute the results. DBXplorer [1] exploits the database schema to find Candidate Networks (join trees) connecting all of the potential tuples that collectively contain the query keywords. DISCOVER [2] also uses the database schema to generate candidate networks. Then both systems leverage Database Management System (DBMS) to evaluate the CNs using SQL join queries and return results to users. IR-Style [3] extends DISCOVER to perform relevance-ranked and top-K keyword queries. The drawback of these approaches is that the execution is usually inefficient because they can access data indirectly using SQL queries on DBMS. The large number of generated join queries lead to high I/O cost.

Schema-free approaches view a database as a graph where nodes represent tuples and edges represent the foreign key relationships between pairs of tuples. The interconnected tuples that contain all or some of input keywords are then identified to answer keyword queries. BANKS [4] employs a backward expanding search algorithm which is based on Dijkstra's single source shortest path algorithm. It expands node clusters by visiting nodes backward until the rooted tree is completed. Traversing the data graph in order to construct the Steiner tree is inefficient because minimum Steiner tree problem is known to be NP-hard [8]. BANKS-II [5] improves the search efficiency of BANKS through Bidirectional expansion technique. The drawback of these methods

is that a data graph must be materialized and kept into memory and this is not suitable for large size databases. In [6, 7] other techniques are presented for pre-computing and materializing the neighborhood for each distinct keyword in the database offline in order to reduce the query processing time. In [10] a polynomial time solution for the minimum Steiner tree problem was introduced. A compact Steiner tree is used to answer keyword queries efficiently. It devises a structure-aware index to materialize structural relationships. Although the methods above facilitate the KSORDB, they still have two main challenges, controlling both the granularity and the size of the index as they are related directly to the database size. The system proposed in this paper belongs to the schema-free approach. It maintains the graph connectivity information in a reachability index in order to overcome the scalability and high memory requirement limitations of this approach.

In [11] a method to support ontology-based semantic matching in RDBMS was presented. This approach makes ontology-based semantic matching available as part of SQL by creating set of new operators, namely ONT_RELATED, ONT_EXPAND, ONT_DISTANCE, and ONT_PATH. It does not support keyword search as users must be aware of these operators to use them when needed. Si-SEEKER [12] provides ontology based semantic search over a schema-based KSORDB system. It is required to annotate all data in the database and then use those annotations to build a semantic index. During query processing, a user keyword query is mapped to concept query then the similarity between this query and the annotated data are calculated to find results. While Si-SEEKER focuses on the improvement of the effectiveness of the results, its semantic searcher is not efficient as its author stated. However, our work provides semantic match capability over schema-free KSORDB system and progressively identifies the query related terms to efficiently generate related results.

3. PROPOSED SYSTEM ARCHITECTURE

This section describes the proposed system architecture. The system has two execution stages: Pre-processing stage and Query processing stage.

Figure 1 shows in details the different components of the system. A detailed illustration of each component is given in the following subsections.

3.1 Pre-Processing Stage

During the pre-processing stage, the system builds some structures that are later used while processing the keyword search query. This stage consists of the Ontology pre-processing module and the Database Pre-processing module.

Ontology pre-processing module In this module, the **Parser** loads and parses the Ontology to construct the **concept graph** that models the ontology as a weighted directed graph where nodes/vertices represent concepts and edges specify the relationships (specifically "is-a" links) between concepts. The edge weight reflects the similarity value between concepts depending on the direction of the edge. Both upward edge (generalization) weight and the downward edge (specialization) weight can be specified by ontology designers or domain experts. Then, the **Annotator** associates each concept with a list of tuples that satisfy the concept and populates the **concept table** with these annotations.

Database Pre-processing Module The underlying database is modeled as a data graph *G (V, E)*, where nodes represent tuples and edges represent foreign - primary key relationships. The data graph can be directed or undirected graph. The directed graph has two types of edges: forward edge *(u, v)* and backward edge *(v, u)*. Accordingly, for each forward edge *(u, v) Є E,* there is its corresponding backward edge *(v, u) Є E* with a different weight. When such differentiation in weights is not necessary, each forward edge and its backward edge are combined into one edge producing an undirected graph. The **Indexer** builds an inverted index that is used to efficiently identify the tuples in the database containing query keywords. For each distinct keyword in the database, it stores a list of tuples that contains the keyword.

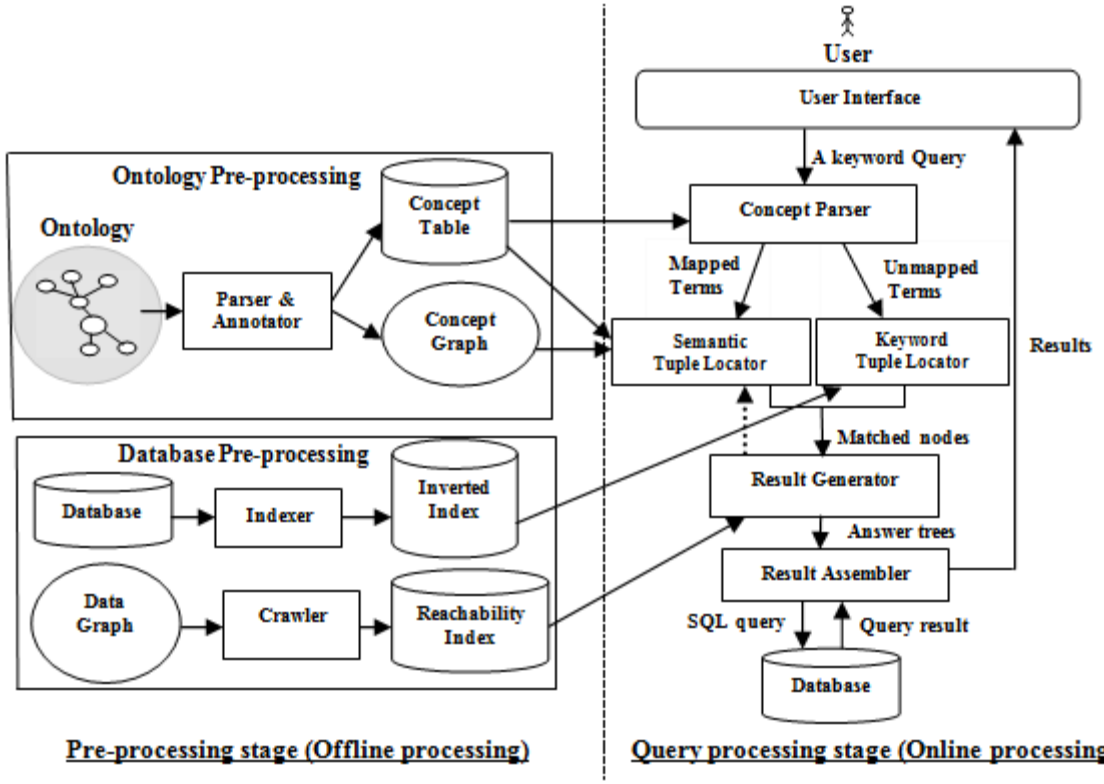

Fig. 1. Proposed System Architecture

The **Crawler** traverses the data graph in a breadth-first manner. The main objective of the crawler is to materialize the neighborhood of the tuples that contain text attributes, called content node, into a reachability index. The neighborhood of each node ***u*** is represented by an entry in the reachability index. Each entry is a list of ***<V,weight,P>*** entries where, ***V*** is a node connected to ***u*** via a path of length ***dmax*** or less, ***weight*** is the weight of the shortest path from ***u*** to ***V***, and ***P*** is the nodes constructing that path. The maximum distance ***dmax*** is provided to restrict the exploration to neighbors that are within a given diameter. Lists are sorted by ***weight***. The crawler iteratively constructs the neighborhoods of content nodes and populates the reachability index. Then, the reachability index is used at the query time to map nodes to their neighborhoods. With the reachability index there is no need to maintain the whole data graph in memory. This is how the proposed system handles the critical issue of working with large and dense data graphs.

3.2 Query Processing Stage

The user enters a query consisting of a set of keywords. The system answers it as follows:

- **Concept Parser.** The concept parser maps the query terms to ontology concepts. It extracts all n-grams from the query [13], sorts them descending by the number of terms, and checks whether the ontology contains a concept that match each of them. If a match is found, the remaining smaller n-grams (covered by this concept) are skipped. Using this strategy, each query term either belongs to *mapped concepts/terms* or doesn't belong to any of them, *unmapped terms*.
- **Semantic Tuple Locator** receives the mapped concepts from the concept parser and exploits the Ontology to progressively return related terms/concepts along with the tuples associated with these terms. For this, the concept table is looked up to find the tuples associated with the mapped concepts. Next, if there was any concept that has no annotation instance (i.e. there was no tuple satisfying this concept) or when the number of the final answers generated by the system is less than what the user needs, the concept graph is traversed starting from nodes that correspond to the mapped concepts to identify the related concepts in a recursive manner based on their degree of similarity to the mapped concepts. Section 4 discusses measuring semantic similarity between concepts and the adopted method.
- **Keyword Tuple Locator** receives unmapped terms/keywords and looks up the inverted index to find tuples that satisfy these terms.
- **Result Generator** receives a set of matched nodes that satisfy each mapped and unmapped term from Semantic Tuple Locator and Keyword Tuple Locator respectively. Given the matched nodes of query terms, the objective of the result generator is to progressively produce the top-k answer trees. An answer tree is a distinct rooted tree containing at least one matched node for each term and there is at most one answer rooted at each node. The weight of an answer tree is the sum of shortest path distances/weights from the root to the matched nodes (leaves) of the tree. Note that the matched nodes of a term may satisfy the term either syntactically or semantically. For this, the result generator does the following: (1) the reachability index is looked up for each term to retrieve the neighborhoods of its matched nodes, called term neighborhood. (2) A cursor is constructed for each term in order to traverse its neighborhood in sorted order (i.e. nodes with smaller weights are accessed first). (3) The cursors are called in a round robin manner in order to identify the Top-k nodes (roots) that has minimum weight sum; in order to identify such nodes efficiently, a top-k algorithm [14] is employed; the pruning technique presented in [4] is used to discard duplicated trees and trees whose root has only one child. (4) If less than K results have been generated then the Semantic Tuple Locator is recursively called in order to retrieve more related results.
- **Result Assembler** receives the IDs of tuples constructing each answer and retrieves the real information of each tuple from the database to deliver the final results to the user.

4. ONTOLOGICAL SEMANTIC SIMILARITY METHODS

Several methods for measuring semantic similarity between terms based on ontology have been proposed in the literature. They can generally be classified into four categories:

- Edge Counting Methods: this type of measures determine the semantic similarity between two terms/concepts based on how close the two terms in the taxonomy are [15].
- Information Content methods: measure the similarity based on how much information the two concepts share [16].

- Feature Based Measures: measure the similarity based on the properties of the concepts [17].
- Hybrid Methods: the similarity in this category is based on combination of previous options [18].

Most semantic similarity methods assign higher similarity value to terms which are close together in the taxonomy. Edge counting methods exploit the structure/taxonomy of the Ontology to evaluate the distance between the ontological nodes corresponding to the concepts being compared. [15] Has presented a simple method to calculate the similarity between concepts based on the shortest path length connecting their corresponding nodes. Thus, the distance between two concepts $\boldsymbol{A}$ and $\boldsymbol{B}$ is computed as:

$$\text{dist(a,b)} = \text{minimal number of edges separating } \mathbf{a} \text{ and } \mathbf{b} \quad (1)$$

Where $\boldsymbol{a}$ and $\boldsymbol{b}$ are the ontological nodes representing $\boldsymbol{A}$ and $\boldsymbol{B}$ concepts respectively. This indicates that the shorter the path, the more similar the terms are. This measure suffers from being symmetric measure (i.e. sim(a,b) = sim(b,a)). An asymmetric similarity measure has proposed in [19]. It argued that while concept inclusion (is-a) intuitively implies strong similarity in the opposite direction of inclusion (specialization), the direction of inclusion (generalization) must contribute with some degree of similarity. Thus, the degree to which $\boldsymbol{a}$ is similar to $\boldsymbol{b}$ is defined as:

$$\mathbf{sim(a,b)} = \boldsymbol{max}_{\boldsymbol{j}=\mathbf{1},\ldots,\boldsymbol{m}} \left\{ \boldsymbol{\delta}^{\boldsymbol{s}(\boldsymbol{P}^{\boldsymbol{j}})} \, \boldsymbol{\gamma}^{\boldsymbol{g}(\boldsymbol{P}^{\boldsymbol{j}})} \right\} \quad (2)$$

Where $\boldsymbol{\delta}$ is the downward (specialization) weight, $\boldsymbol{\gamma}$ is the upward (generalization) weight, $\boldsymbol{s}(\boldsymbol{P}^{\boldsymbol{j}})$ and $\gamma(P^j)$ are number of specializations and generalizations along the path j respectively. This indicates that the similarity between two concepts $\boldsymbol{a}$ and $\boldsymbol{b}$ is the maximal product of weights along the paths between $\boldsymbol{a}$ and $\boldsymbol{b}$.

In order to progressively expand the query concepts with related concepts, the Semantic Tuple Locator employs function (2) as a similarity function. Thus starting with nodes corresponding to the query concepts, the concepts with highest similarity are retrieved first followed by those with lower values.

5. PERFORMANCE EXPECTATIONS

Through implementing the proposed system, we expect the following performance enhancements:

Storage requirements reduction The proposed reachability index is expected to reduce the space requirements compared to existing techniques [7, 10]. This is because the materialization strategy employed by our system materializes the neighborhood of each content node only once, but other systems redundantly materialize these data, a content node's neighborhood, either for each keyword in the content node (as in [7]) or for each keyword in the neighbor nodes that does not exist in the content node (as in [10]). Therefore, while the granularity and the size of the indexes maintained by other systems are affected greatly by the number of distinct keywords in each tuple, the proposed reachability index is not affected by this factor.

Quality of results While existing schema free KSORDB systems generate structures that connect the nodes that exactly (syntactically) contain the query terms (we call such results exact or direct

results), the proposed system also find semantic/related results that may connect nodes that do not contain the exact query terms but related terms. Thus, even though existing systems generate results with high precision, their recall is relatively small, as stated in [12]. The proposed system exploits the structure of domain ontology to progressively return query related terms that can be used to retrieve more relevant answers and hence increasing the effectiveness in terms of the recall rate of results.

Performance In order to efficiently identify the tuples containing terms that are related to a query terms and may contribute in the final results, the proposed semantic tuple locator exploit the structure of domain Ontology to progressively return related terms and pass the tuples associated with these terms to the Result generator. Compared to the semantic searcher of Si-SEEKER [12], the proposed technique is expected to reduce the time required to identify these tuples. This is because the Si-SEEKER semantic searcher exploits domain ontology structure to calculate the semantic similarity between the query and all tuples in the database and then picks tuples with similarity value greater than a given threshold. Therefore it takes linear time in terms of the total number of database tuples no matter the number of needed results. On the other hand, the semantic tuple locator exploits the ontology structure to progressively identify query related terms and then the tuples associated with these terms. Accordingly, the progressive manner adopted by the semantic tuple locator generates only the highly related tuples needed to generate the final results without being affected by the number of tuples in database.

Currently, we are implementing and performing experiments using restbase[1] data set. A Cuisine ontology (specifically is-a hierarchy) is used as a domain specific knowledge.

REFERENCES

[1] S. Agrawal, S. Chaudhuri, and G. Das: DBXplorer: A System for keyword Search over Relational Databases. In: ICDE, pp. 5-16 (2002)
[2] V. Hristidis and Y. Papakonstantinou: DISCOVER: Keyword search in relational data-bases. In: Proc. 28th Int. Conf. On Very Large Data Bases, pp. 670–681 (2002)
[3] V. Hristidis, L. Gravano, Y. Papakonstantinou: Efficient IR-Style Keyword Search over Relational Databases. In: VLDB, pp. 850-861 (2003)
[4] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan: Keyword searching and browsing in databases using BANKS. In : Proc. 18th Int. Conf. on Data Engineering, pp. 431–440 (2002)
[5] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar: Bidi-rectional expansion for keyword search on graph databases. In VLDB, pp. 505-516 (2005)
[6] H. He, H. Wang, J. Yang, and P. Yu: BLINKS: Ranked keyword searches on graphs. In: Proceedings of the 2007 ACM SIGMOD international conference on Management of data, pp. 305-316 (2007)
[7] Guoliang Li, Jianhua Feng, Feng Lin, Lizhu Zhou: Progressive Ranking for Efficient Keyword Search over Relational Databases. Proceedings of the 25th British national con-ference on Databases: Sharing Data, Information and Knowledge, pp. 193 - 197 (2008)
[8] Garey, M.R., Johnson, D.S.: The rectilinear Steiner tree problem in NP complete. SIAM J. Appl. Math. 32, pp. 826–834 (1977)
[9] Gruber: Toward Principles for the Design of Ontologies Used for Knowledge Sharing. In IJHCS, pp. 907-928 (1994)
[10] Li, Guoliang; Feng, Jianhua; Zhou, Xiaofang; Wang, Jianyong: Providing built-in keyword search capabilities in RDBMS. In VLDB, pp. 1-19 (2011)

[1] ftp://ftp.cs.utexas.edu/pub/mooney/nl-ilp-data/restsystem/restbase

[11] S. Das, E.I. Chong, G. Eadon, J. Srinivasan. Supporting Ontology-Based Semantic match-ing in RDBMS. On VLDB, pp. 1054-1065 (2004)

[12] Jun Zhang, Zhaohui Peng, Shan Wang, Huijing Nie: Si-SEEKER: Ontology-Based Se-mantic Search over Databases. In: Knowledge Science, Engineering and Management (2006) 4092, pp. 599-611 (2006)

[13] Edgar Meij , Marc Bron , Laura Hollink , Bouke Huurnink , Maarten de Rijke: Mapping queries to the Linking Open Data cloud: A case study using DBpedia, Web Semantics: Science, Services and Agents on the World Wide Web, v.9 n.4, pp. 418-433 (2011)

[14] Nikos Mamoulis, Kit Hung Cheng, Man Lung Yiu , David W. Cheung: Efficient Aggre-gation of Ranked Inputs. In: Proceedings of the 22nd International Conference on Data Engineering, page 72 (2006)

[15] Rada, R., Mili, H., Bichnell, E., & Blettner, M.: Development and application of a metric on semantic nets. IEEE Transactions on Systems, Man, and Cybernetics, pp. 17–30 (1989)

[16] Resnik, P.: Using Information Content to Evalutate Semantic Similarity in a Taxonomy. In: IJCAI, pp. 448–453 (1995)

[17] Tversky, A.: Features of similarity. Psycological Review, 84, pp. 327–352 (1977)

[18] Rodriguez, M. A., & Egenhofer, M. J. Determining semantic similarity among entity classes from different ontologies. IEEE Transactions on Knowledge and Data Engineering, 15, pp. 442–456 (2003)

[19] Bulskov, H.; Knappe, R.; Andreasen, T.: On measuring similarity for conceptual querying. In: 5th international conference, FQAS 2002, Copenhagen, Denmark, pp. 100-111 (2002)