# MeanKS: Meaningful Keyword Search in Relational Databases with Complex Schema

Mehdi Kargar[1], Aijun An[1], Nick Cercone[1], Parke Godfrey[1], Jaroslaw Szlichta*[2] and Xiaohui Yu[1]
[1] Department of Computer Science and Engineering, York University
{kargar,aan,nick,godfrey,}@cse.yorku.ca and xhyu@yorku.ca
[2] University of Ontario Institute of Technology
jaroslaw.szlichta@uoit.ca

## ABSTRACT

Keyword search in relational databases was introduced in the last decade to assist users who are not familiar with a query language, the schema of the database, or the content of the data. An answer is a join tree of tuples that contains the query keywords. When searching a database with a complex schema, there are potentially many answers to the query. Therefore, ranking answers based on their relevance is crucial in this context. Prior work has addressed relevance based on the size of the answer or the IR scores of the tuples. However, this is not sufficient when searching a complex schema.

We demonstrate MeanKS, a new system for meaningful keyword search over relational databases. The system first captures the user's interest by determining the roles of the keywords. Then, it uses schema-based ranking to rank join trees that cover the keyword roles. This uses the relevance of relations and foreign-key relationships in the schema over the information content of the database. In the demonstration, attendees can execute queries against the TPC-E warehouse and compare the proposed measures against a gold standard derived from a real workload over TPC-E to test the effectiveness of our methods.

## 1. INTRODUCTION

Keyword search, a well known mechanism for retrieving relevant documents in information retrieval, has recently been used for extracting information from relational databases. Users usually do not have sufficient knowledge about the structure of data nor query languages such as SQL [2, 3]. Keyword search over databases can bridge this gap. An answer to the query is a set of tuples from the database that *cover* the keywords of the query, and a natural *structure* —a tree from the database's schema— that spans those tuples. Previous work restricts answers over *minimal* trees [2, 1], meaning there is no answer over a sub-tree of the tree in question. However, there is another aspect of an answer: relevance to the query. By restricting answers to a minimal structure, some relevant answers may be missed.

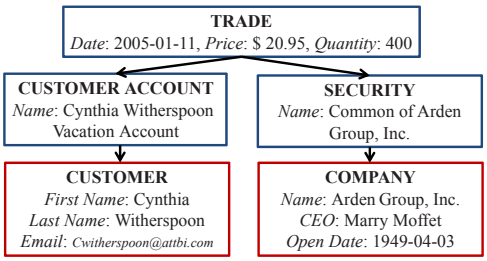


**Figure 1: A non-minimal answer by DISCOVER [2].**

Consider the keyword query *"Cynthia Arden"* over the TPC-E[1] schema. The TPC-E benchmark simulates the *online transaction processing* (OLTP) workload of a brokerage firm.[2] The keywords *Cynthia* and *Arden* may appear in different relations. Each could refer to the name of a *customer*, *broker*, *company*, CEO of a *company*, or be in the title of a *news item*. Of course, each of these different relations for *Cynthia* and for *Arden* potentially lead to rather different answers. For example, Figure 1 shows an answer for the keyword query *"Cynthia Arden"* on TPC-E, which says that a customer *Cynthia* buys the stocks of a company named *Arden Group*. This is an interesting and relevant relationship between *Cynthia* and *Arden* assuming, the user wants to find out the relationship between customer *Cynthia* and company *Arden Group*. However, such an answer would be missed if we restrict the answer to be structurally minimal while covering all the query keywords (This happens in the previous works such as DISCOVER [2].) This is because *Cythia* also appears in an intermediate node of the tree and thus the *customer* node is pruned. This also arises for *Arden*.

Another issue in keyword search is to score answers for *relevance*. The relevance of an answer depends on many factors: the tuples in the answer and how the keywords appear in them; the bridging tuples that do not contain keywords; and the join tree. How to measure the value of each of these is open to question. Prior work has addressed relevance. In [2], a simplistic solution of scoring relevance as the reciprocal of the number of edges in an answer's tree was proposed. This heuristic assumes that fewer joins involved mean the tuples are related more closely. This approach might work for a small and simple schema, but it fails to return relevant

*Szlichta is also a Fellow at IBM Centre for Advanced Studies in Toronto.


*SIGMOD'14,* June 22–27, 2014, Snowbird, UT, USA.

http://dx.doi.org/10.1145/2588555.2594533.

[1]http://www.tpc.org/tpce/
[2]The TPC-E has a reasonably complex schema and contains 33 tables.

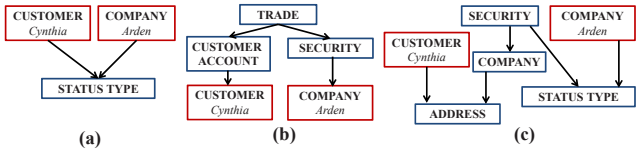


**Figure 2: Three possible trees (i.e., MJNSs) for the query *"Arden:company Cynthia:customer"*.**

answers when the schema is large and complex. Consider the query {*Arden Cynthia*} over the TPC-E schema. Also, assume that *Arden* refers to a company name and *Cynthia* refers to a customer name. Figure 2 shows three possible join trees of different sizes that could produce answers that connect the company *Arden* and the customer *Cynthia*.

If we rank the trees according to their size (i.e., the number of edges or nodes), the first tree (a) which connects a customer and a company when they have the same status gets the highest rank. However, based on the TPC-E schema description, this is not a strong relationship; *status type* is a dimension table which is connected to six tables in the schema and stores the status value for other entities (such as companies, customers and trades). An answer derived from this tree would say that both *Arden* and *Cynthia* are *active*. Looking at the *customer* and a *company* tables in TPC-E, it turns out that all the customers and companies have the *Active* status. Therefore, any given customer and any given company in the TPC-E database share the same status through the *status type* table. Thus, the first found relationship (a) between *Arden* and *Cynthia* is, in fact, not that interesting or relevant. The second tree (b) states that company *Arden*'s stock is traded by customer *Cynthia*. This is one of the strongest relations between a customer and a company and is related to the purpose of the TPC-E schema. In addition, it usually produces few results since trading the stocks of one specific company by one specific customer is not common. The third tree (c) is the largest and therefore it is not straightforward to interpret. This tree states that the company *Arden* has the same status as a security which is related to a company that has the same address as customer *Cynthia*. Two relations, *address* and *status type*, that are involved in this tree makes it less meaningful and more difficult to interpret for the user.

In [1], the authors take a different tact: they apply information retrieval (IR) measures to the answers to determine an individual measure per answer. We believe that for keyword search in relational databases, the relevance of the tree from which answers derive is paramount. We hypothesize that relevance as measured by adapted IR techniques is much less effective. For example, for the trees in Figure 2, assume that the IR scores of the tuples that contain keywords are the same.[3] Applying the ranking technique of [1], the first tree (a) gets the highest score since it is the smallest one. As we discuss previously, this is not a relevant answer for the given query. In our system, IR scores are used as a secondary sorting key to rank the final answers with the same schema ranking (See Section 3.2 for more details.)

In this demonstration, we showcase a new system for keyword search in relational databases (MeanKS)[4]. The unique features of our system are as follows.

1. The system identifies the database entities that are potentially interesting to the user based on the query keywords the user provides, and allows the user to specify their interests through a user interface.

2. The system returns the top minimal connected tuples that not only contain the query keywords but also cover the user interests (referred to as keyword roles in our system).

3. The system uses schema-based ranking that ranks the answers based on the minimal join trees that cover the keyword roles.

4. The system provides alternative relevance measures for join trees, which consider the importance of nodes, edges or combinations of them.

## 2. PROBLEM STATEMENT

A relational database schema consists of a set of $n$ relation schemas, denoted as $\{R_1, R_2, \ldots, R_n\}$, where each $R_i$ is described by a set of attributes. Two relation schemas, $R_i$ and $R_j$, may be related by a *foreign key relationship*, denoted as $R_i \leftarrow R_j$, where the primary key of $R_i$ is referenced by a foreign key of $R_j$. Thus, a relational database schema can be considered as a graph $G = \langle V, E \rangle$ where nodes are relation schemas and edges represent the foreign key relationships. A relational database is an instance of a relational database schema $G$. It consists of a set of relations, where each relation $r(R_i)$ is a set of tuples conforming to the relation schema $R_i$ in $G$. Given a relational database $D$ and a set of $l$ ($\geq 2$) keywords ($Q = \{k_1, k_2, \ldots, k_l\}$), the problem of keyword search over $D$ is to find a set of tuples that are connected via foreign key relationships and cover all the keywords in $Q$.

In our framework, the role of a query keyword is identified first. For each query keyword, we first find the list of columns (and relations) that contain the keyword using an inverted index or the built-in support for full-text keyword search in the RDBMS of choice, and then a relation is chosen through interaction with the user from a list as the ***role of the keyword***. With specified keyword roles, our algorithm searches for answers that are defined as follows.

DEFINITION 1. ***Minimal Joining Network of Tuples Covering Roles***. *Given a database $D$ with schema graph $G$ and a query $Q$ containing a set of keywords $\{k_1, k_2, \ldots, k_l\}$ and their respective roles $\{r_1, r_2, \ldots, r_l\}$ (where $r_i$'s are relations in $D$), a minimal joining network of tuples for query $Q$ is a tree $T$ of tuples that satisfy the following conditions:*

***1) Joinable**. For each edge $(t_i, t_j)$ in $T$, where $t_i \in r(R_i)$ and $t_j \in r(R_j)$, there is an edge $R_i \leftarrow R_j$ or $R_i \rightarrow R_j$ in $G$ and $t_i \bowtie t_j \in r(R_i) \bowtie r(R_j)$.*

***2) Role and keyword covering**. For each query keyword role $r_i$, there exists a node $t_j$ in $T$ such that $t_j \in r_i$ and $t_j$ contains keyword $k_i$.*

***3) Minimal**. If a tuple in $T$ is removed, $T$ is either not joinable or does not cover all the roles or all the keywords.*

For brevity, we refer to the answer defined in Definition 1 as a ***final answer***. A final answer covering roles *customer* and *company* is shown in Figure 1. Given a database, there may be many final answers to a query. Instead of producing all the answers which may overwhelm the user, the goal of our algorithm is to produce the most meaningful final answers. The final answers defined above can be generated

[3]This happens when *Cynthia* is the first name of a customer and the related column contains only "*Cynthia*"

[4]Demo available at http://graph.cse.yorku.ca:8080/meanks

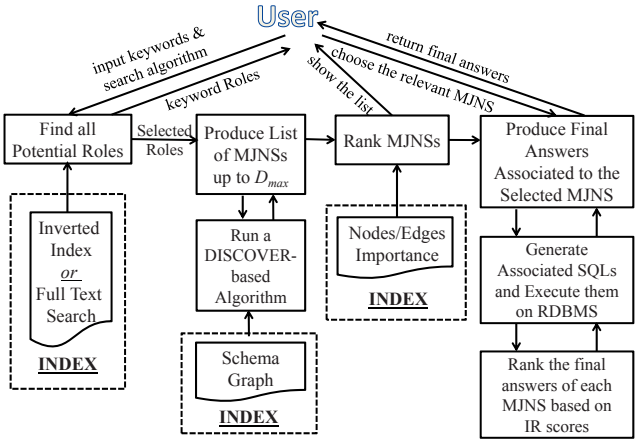


**Figure 3: System Architecture.**

through a sequence of join operations on the database. To generate such answers, we first generate the *minimal joining networks of schemas (MJNSs)* that represent the join operations for producing the final answers. Next, we define MJNS, and present our method for generating them.

DEFINITION 2. ***Minimal Joining Network of Schemas (MJNS)****: Given a database $D$ with schema graph $G$ and a set $r$ of query keyword roles $\{r_1, r_2, \ldots, r_l\}$ (where $r_i$'s are relations in $D$), a minimal joining network of schemas that cover $r$ is a tree $T$ of relation schemas in $G$ that satisfy the following conditions.* ***1) Joinable****. Each edge in $T$ is an edge in $G$.* ***2) Role covering****. For each query keyword role $r_i$, its schema is in $T$.* ***3) Minimal****. If a relation schema in $T$ is removed, $T$ is either not joinable or does not cover all the roles in $r$.*

Three possible MJNSs for the query *"Arden:company Cynthia:customer"* over TPC-E schema is shown in Figure 2. To generate MJNSs, we use a breadth-first search algorithm similar to the candidate network (CN) generator of DISCOVER [2]. Our algorithm starts with a role schema as an initial tree $T$ and extends $T$ with a relation schema in $G$ that has a foreign key relationship with a node in $T$. The expansion of $T$ stops once the schemas of all the roles are covered in $T$. To avoid generating duplicate trees, each generated tree is assigned an ID based on tree isomorphism during the execution of the algorithm. The ID of a tree is checked against the existing IDs that are generated so far and the current tree is accepted if it is not generated previously.

After the MJNSs are generated, final answers can be produced by creating an execution plan to evaluate the MJNSs. Since many final answers can be generated for a query but some answers may not be interesting, we aim at producing the most interesting final answers. To achieve this, we first limit the number of nodes in an MJNS (which is a strategy taken by DISCOVER as well). This limits the size of final answers too.

# 3. SYSTEM ARCHITECTURE

The architecture of our system for finding meaningful answers is shown in Figure 3. It consists of four functional components and three data components. Below we first describe the data components and then describe the functional components. The search strategy and ranking algorithms are discussed at the end of this section.

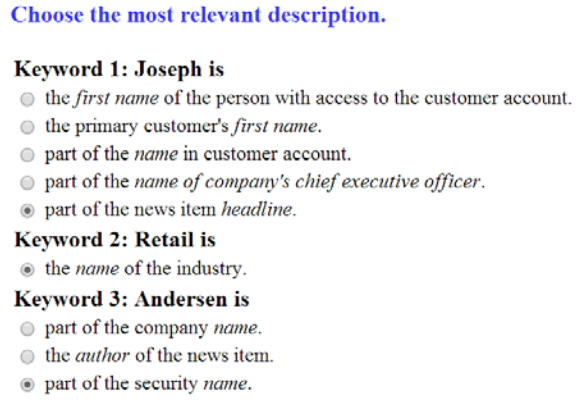


**Figure 4: Role selection by the user for query "Joseph Retail Andersen".**

## 3.1 Data Components

For a given relational database and its schema, the following three data structures are pre-built to facilitate the search process.

**Inverted Index:** For each keyword in the database, the columns with its associated table that contain the keyword are stored in this index, which is maintained and accessed through a table in RDBMS. This index is used to find the roles of each keyword. We can alternatively use the built-in full text search of the RDBMS.

**Schema Graph:** In order to produce MJNSs, we need to represent the database schema in a graph. In our system, we use the neighbor index to implement the graph. For each node, the neighbors of the node are stored in the index. The relations between the tables (i.e., the foreign key relationships) are stored in this index as well.

**Node/Edge Importance:** As discussed before, our ranking strategy works based on the importance of nodes and edges of the database schema. In order to improve the performance, the scores of the nodes and edges are calculated offline and is stored in a table in the RDBMS.[5] This information is later used for ranking the list of MJNSs.

## 3.2 Functional Components

After receiving the query keywords from the user, for each keyword, our system finds the columns that contain one of the keywords using the inverted index. The list of descriptions of these columns for each keyword is shown to the user so s/he may choose the most relevant description. Such descriptions tell the meanings of the attributes, and can be provided by people (such as DBAs) familiar with the database schema.[6] Based on the user selection, the role (i.e., the relevant database entity) of each keyword is identified. The role selection page is shown in Figure 4.

After finding the role of each keyword, these roles are passed to an engine to produce the list of MJNSs (up to the maximum size, $D_{max}$). The engine uses the schema graph to produce the list of MJNSs. In the next step, this list is sorted according to the chosen MJNS ranking measure. The sorted list is then shown to the user for selecting the appropriate MJNS. Three MJNSs with maximum size of six (among 42 possible MJNSs) for the query *"Arden:company Cynthia:customer"* over TPC-E schema are shown in Figure 2. For example, if the *IF* method (to be described later) is

[5]The scores should be updated periodically if the database content changes.

[6]In our experiments on the TPC-E database, the short attribute descriptions are taken from the TPC-E document.

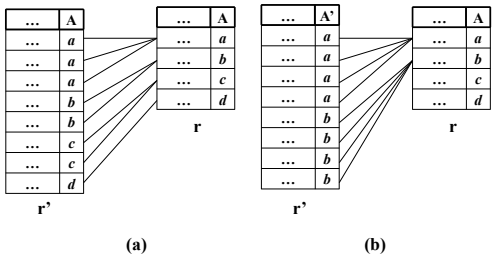


**Figure 5: Two different instances of the foreign key connections between tables $r$ and $r'$**

used for ranking, the second MJNS (b), will get the highest rank among the others. For each selected MJNS, the associated SQL statement is generated and executed against the RDBMS. The final answers from this selected MJNS are then ranked by their IR scores. A final answer, associated to the second MJNS in Figure 2 (b), is shown in Figure 1.

## 3.3 Ranking Strategies

We propose and use several methods for ranking minimal joining networks of schemas (MJNSs). The proposed methods work just based on the database schema and its given instance, assuming that no extra information (e.g., query logs or inheritance relationships among the tables) is available. The methods use information-theoretic measures to evaluate the importance of relations and/or edges in the given database, and rank the MJNSs based on the importance of the relations and/or edges in an MJNS. We classify the proposed methods into three categories: (1) ranking based on the importance of the edges in an MJNS; (2) ranking based on the importance of the nodes in an MJNS; and (3) ranking based on the importance of both nodes and edges in an MJNS (i.e., a hybrid method). We found in [4] that one of the proposed edge based ranking methods ($IF$) outperforms other methods[7]. (Due to space limit, we only present one ranking method here; however, all are available in the system and for demonstration.) The details of our procedure for ranking MJNSs using other ranking strategies are described in Kargar et al., [4].

We use the average importance score of the tables and edges associated with these non-role relation schemas to compute an importance score for an MJNS. To have a ground truth, we construct a gold standard for relevance of answers from a transaction log of the TPC-E database. This is used to evaluate the effectiveness of our measures which do not require such external information.

**Edge-based Ranking, Instantiation Fraction (IF):** Intuitively, edge strength can be measured by the fraction of the join key values being instantiated. The more fraction of primary key values are instantiated, the more important the edge is. Unlike methods for summarizing database schema in [5], we do not assume that by increasing the number of connections between two tables, the importance of the edge decreases. Therefore, we propose the following measure, called *instantiation fraction* (IF), to quantify the importance of an edge based on the fractions of instantiated key values.

$$ST_{IF}(r.A, r'.A') = \frac{N^{inst}_{r.A}}{N_r} \times \frac{N^{inst}_{r'.A'}}{N_{r'}}$$

where $N^{inst}_{r.A}$ is the number of tuples in $r$ that instantiates the edge between $r.A$ and $r'.A'$, and $N_r$ is the total number of tuples of table $r$. $ST_{IF}(r.A, r'.A')$ of the the left and right instances in Figure 5 is equal to 1 and 0.5, respectively. We believe the left instance represents a stronger association between $r$ and $r'$. This is the reason the first MJNS (a) in Figure 2 gets a lower rank than the second MJNS (b). The *status type* table has five different statuses (i.e., Completed, Active, Submitted, Pending, and Canceled). Neither the *customer* table nor the *company* table instantiate all of the statues of the *status type* table.

[7]We evaluated different methods through a user study and by comparing them to the gold standard.

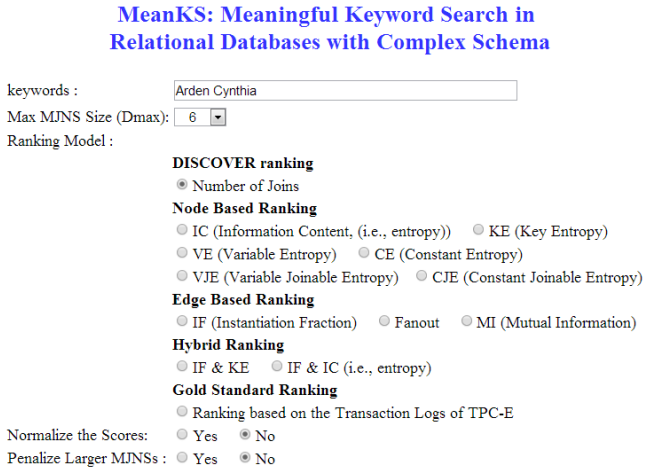


**Figure 6: First page of the system with parameters.**

# 4. DEMONSTRATION PLAN

We developed a web-based browsing interface using Java to support interactive keyword search in a relational database (i.e., the TPC-E warehouse). The first page of the interface is shown in Figure 6, where the user can specify query keywords and other parameters of the system (such as the ranking method, whether to penalize larger MJNSs and to normalize the scores) (See [4] for more details.) The second page of the system is for choosing the right roles (Figure 4). Then, the ranked list of MJNSs, based on the selected ranking method, is shown to the user (Figure 2). The last step for the user is to choose the relevant MJNS. The final answers associated to the selected MJNS, which is a list of trees of tuples ranked by their IR scores, are shown in the last page (one final answer is shown in Figure 1). In the demonstration, users are able to try different search methods and compare the accuracies of them against a gold standard.

# 5. REFERENCES

[1] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient ir-style keyword search over relational databases. In *Proc. of VLDB'03*, 2003.

[2] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *Proc. of VLDB'02*, 2002.

[3] M. Kargar and A. An. Keyword search in graphs: Finding r-cliques. In *Proc. of VLDB'11*, 2011.

[4] M. Kargar, A. An, P. Godfrey, J. Szlichta, and X. Yu. Meaningful Keyword Search in RDBMS. Technical report, 2013. www.cse.yorku.ca/techreports/2013.

[5] X. Yang, C. M. Procopiuc, and D. Srivastava. Summarizing relational databases. In *Proc. of VLDB'09*, 2009.