Deanship of Graduate Studies
Al-Quds University

System for Top-k Keyword Search processing over relational databases using semantics

Samia Taha Hussein Abdulhay

M.SC. Thesis

Jerusalem – Palestine

1431 / 2010

System for Top-k Keyword Search processing over relational databases using semantics

Prepared By:
Samia Taha Hussein Abdulhay

B.Sc. : Computer Science, Al-Quds University, Palestine

Supervisor :
Dr. Rashid Jayousi

A thesis Submitted in Partial Fulfillment of Requirements for the Master Degree of Computer Science from Computer Science Department of Al-Quds University.

1431 / 2010

Deanship of Graduate Studies
Al-Quds University

Thesis Approval

System for Top-k Keyword Search processing over relational databases using semantics

Prepared By: Samia Taha Hussein Abdulhay
Registration No: 20510194

Supervisor: Dr. Rashid Jayousi

Master thesis submitted and accepted Date: 15/5/2010

The names and signatures of examining committee members are follows:

1. **Head of Committee: Dr. Rashid Jayousi Signiture:…………..**
2. **Internal Examiner: Dr. Nedal Kafri Signiture:…………..**
3. **External Examiner: Dr. Yousef Abuzir Signiture:…………..**

Jerusalem – Palestine

1431/ 2010

Dedication

I would like to dedicate this work to my family, who supported me in all phases of this thesis, especially to my lovely mother, for her support, care and love, and to all the people who help me to overcome difficulties.

Samia taha Hussein Abdulhay

Declaration

I certify that this thesis submitted for the degree of Master of Computer Science, is the result of my own research, except where otherwise acknowledged, and that this thesis (or any part of the same) has not been submitted for a higher degree to any other university or institution.

Signed …………………

Samia Taha Hussein Abdulhay

Date: 15/5 /2010

Acknowledgements

Grateful thanks for God for giving me the patience and power to complete this work.

I sincerely thank Al-Quds University for giving me the opportunity to study the M.Sc., and its efforts and help throughout this study.

I am deeply grateful, despite the inability of these words to express my thanks to my supervisor, Dr. Rashid Jayousi, for his fruitful discussion, a valuable guidance, continuous support, kindness, and allowing me a lot of his time.

Special thanks to the Library department of Al-Quds University for allowing me to use the Library database to conduct the experiments.

Last, my endless thanks to my mother for her never ending enthusiasm, and to my family for their encouragement and support.

Finally, to the soul of my father…..

Abstract

A variety of keyword search systems over relational database are widely known. Generally such systems do not take into account what is meant from the search query, the user type a list of keywords in the search query, and then the search system retrieve all the results (a set of related records) that contains this keywords, which leads to a high irrelevant results at first top-k. In order to improve the relevancy of such results, this thesis proposed a simple keyword search technique that can help ordinary users to be more specific in expressing their needs.

This can be done by adding some schema information (e.g., table name, field name), which can be used as semantics to the searching keywords. This thesis presents ***Ssearch*** system that is designed to handle the proposed idea.

The researcher has conducted several experiments that use the Library database of Al-Quds University. The experimental results showed that ***Ssearch*** adds a significant improvement in terms of relevancy with acceptable overhead time when compare it with an existing approach.

الملخص

يوجد اليوم عدة انظمة تدعم عملية البحث باستخدام الكلمات المفتاحية (Keyword Search) على قواعد البيانات العلائقية، وقد انتشرت بشكل واسع، وذلك لسهولة استخدامها من قبل المستخدم العادي، ولكن عملية البحث باستخدام الكلمات المفتاحية لا تأخذ بعين الاعتبارالمعنى الذي يقصده المستخدم من عملية البحث، مما يتسبب ذلك في ارجاع عدد كبير من النتائج في الصفحات الاولى ليست ذات علاقة بما يقصده المستخدم، على سبيل المثال، لو اراد المستخدم ان يبحث عن الكتب التي قام بتأليفها المؤلف طه حسين فان عملية البحث ممكن ان تعطيك ليس فقط الكتب التي الفها المؤلف طه حسين فحسب وانما الكتب التي تتحدث عن طه حسين وبيانات المستخدمين الذين يحملون نفس الاسم ايضا، لذلك اقترحت الباحثة في هذه الاطروحة فكرة جديدة لتحسين دقة النتائج التي تظهر في الصفحات الاولى، وذلك باعطاء المستخدم مرونة اكثر للتعبير عن احتياجاته بشكل ادق وذلك بالاستفادة من بعض المعلومات المتوفرة في بنية نظام قواعد البيانات مثل اسم الجدول او اسم الحقل لاسخدامه كمعنى للكلمات المفتاحية التي يستخدمها المستخدم في البحث، وبالتالي مساعدة نظام الحاسوب على تمييز النتائج التي لها صلة بما يعنيه المستخدم بشكل اكبر. لذلك تم اقتراح لغة استعلام بسيطة تستفيد من هذه المعلومات لتحسين دقة النتائج التي تظهر اولا.

لقد تم اجراء عدة تجارب باستخدام قاعدة بيانات مكتبة جامعة القدس، وقد اثبتت هذه التجارب تحسنا ملحوظا في دقة نتائج البحث مع زيادة طفيفة ومقبولة في الوقت اللازم لبدء اخراج هذه النتائج للمستخدم في النظام المقترح ***Ssearch*** بالمقارنة مع نظام اخر قائم.

Table of Contents

List of Figures

List of Tables

Chapter One: Introduction

1.1 Introduction

Keyword search querying has emerged as one of the most effective paradigms for information discovery, especially over relational database. One of the key advantages of keyword search querying is its simplicity – users do not have to learn a complex query language, and can issue queries without any prior knowledge about the structure of the underlying data. Since the keyword search query interface is very flexible, queries may not always be precise and can potentially return a large number of query results, especially in a large amount of data stored within the database. Consequently an important requirement for keyword search is to rank the query results so that the most relevant results appear first.

This thesis proposes a new system where the semantics are a part of the search query which allows the user to be more precise in formulating the query to improve the relevancy of the query results.

This thesis starts by illustrating the previous related work in chapter 2. Chapter 3 provides the needed background about different information retrieval approaches over databases. An existing Approach (***Discover)*** is presented in chapter 4. Then in chapter 5 the proposed system (***Ssearch***) is described in detail. The experimental design and results analysis are in chapter 6. Finally, conclusion and future work are in chapter 7.

1.2 History of Keyword search

Keyword search appears first as a tool for information retrieval over internet where the search engines provide keyword search on top of sets of documents, when a set of keywords is provided by the user, the search engine returns all documents that are associated with these keywords.

Excite introduced the concepts of keyword searching over internet, it was launched in February 1993 by Stanford students and was then called Architext. They had the idea of using statistical analysis of word relationships to make searching more efficient. They were soon funded, and in mid 1993 they released copies of their search software for use on web sites [http://www.searchenginehistory.com].

Today, the dominant of keyword search as a tool for information retrieval over the internet and its simplicity in use, and on the other hand the huge amount of information stored in relational databases that is not well supported for information discovery, lead the researchers to apply keyword search technique as a tool for extracting information from relational database.

The first framework for keyword search on databases is presented in [Masermann, 2000]. The main limitation of this work is that all keywords must be contained in the same tuple. That is, the relationships between tuples from different relations are not taken into consideration. In the next section we will present several keywords search methods over DB.

1.3 Related Work

Several systems apply Keyword Searching on a relational DBMS such as BANKS [Aditya, 2002] which creates a *data graph*, where each node represents a tuple, and edges connect tuples that can be joined (e.g., according to Primary Key - Foreign Key relationships). A Keyword searching query is processed by a graph traversal that searches for *connection trees* containing the query keywords. A connection tree is a Steiner tree in which every leaf node corresponds to a record containing at least one query keyword. Internal nodes represent tuples that connect the leaf records and may include no query keywords. DBXplorer [Agrawal, 2002] and ***Discover*** [Hristidis, 2002] use a higher level of representation - *candidate networks* created from the schema of the database by join operations. The systems use the candidate networks to generate operator trees for evaluating the query. Both DBXplorer and ***Discover*** rank results based exclusively on the distance of the tuples containing the query keywords, whereas [Hristidis, 2003] utilize state-of-the-art IR measures to calculate scores, all of the mentioned systems focus on keyword based searching over a single DBMS, recently [Yu, 2007] and [Vu, 2008] concern on how to support keyword searching over multiple DBMSs, by developing effective ranking methods to select the most useful databases for a given keyword query.

1.4 Problems and Objectives

All search systems that have been mentioned in the previous section, focus on how to generate the graph, extracting the subtree results and display the results using different ranking techniques to improve the relevancy, but no one of them take into account the semantic of the query! This could lead to retrieve irrelevant results at first top-k, especially; if the amount of data stored in the database is large (the precise of the top-k results decreases as the amount of data stored in the database increases [Hristidis, 2002]).

This thesis aims to investigate new technique to improve the search results by utilizing some schema information (table name, column name) as semantic to the related search term (keyword).

After reviewing the literature, we noticed that [Cohen, 2003] developed syntax for search queries over XML that is suitable for naïve user, which allows the user to specify how keywords are related to tags by using the tag name as a semantic to the related keyword.

In our model, we adapt the idea of the query syntax mentioned above for a different environment (keyword searching over relational database) where the users should have a query expression power comparable with database queries, while the language should be kept simple enough to be usable by ordinary user. The users should not be compelled to have inside knowledge of the data structure or schema information of the data they search in, yet once they know or discover it they should be able to take advantage of it.

1.5 Motivation for this thesis

We are motivated by this work for several reasons:

1. Increasing amount of data stored in databases, which leads to decrease the percentage of relevant results that must appear at first.
2. Keyword search is the dominant information discovery method in documents.
3. The simplicity of the keyword search querying (especially for ordinary users).

1.6 Contribution in the field

We believe that our contribution is on the following points:

1. Improve the relevancy of query results that appear at first top-k by adding some semantics to the keywords in the search query.
2. Describing the algorithms needed to build keyword search system, which can support semantics.
3. Clarify the keyword search system workflow.

Chapter Two: Literature Review

2.1 Introduction

Query using keywords is the most widely used form of querying today. While keyword searching is widely used to search documents on the Web, querying of databases currently relies on complex query languages that are inappropriate for ordinary end-users, since they are complex and hard to learn. Even languages, such as QBE, that have been targeted at relatively inexperienced users require the user to be aware of the database schema, which is not appropriate for casual users of an information system. Given the popularity of keyword search, and the increasing use of databases as the back end for data published on the Web, the need for querying databases using keywords is being increasingly felt. One key problem in applying document or web keyword search techniques to databases is that information related to a single answer to a keyword query may be split across multiple tuples in deferent relations.

Survey of work on keyword querying in relational databases like Banks [Aditya, 2002], DBXplorer [Agrawal, 2002], Discover [Hristidis, 2002] etc. along with the comparison between deferent methods is presented in the following sections.[

2.2 Graph Model of Relational Databases

With evolution of different techniques in this area of research, a uniform model has emerged for representing relational databases as a graph with the tuples in the database mapping to nodes and cross references (such as foreign key and other forms of references) between tuples mapping to edges connecting these nodes. The graph model may be used in keyword search as follows:

Each tuple in the database is modeled as a node in the directed graph and each foreign key-primary key link as an edge between the corresponding tuples. This can be easily extended to other type of connections; for example, it can be extended to include edges corresponding to inclusion dependencies, where the values in the referencing column of the referencing table are contained in the referred column of the referred table but the referred column need not be a key of the referred table. Keywords in a given query activate some nodes. The answer to the query is defined to be a subgraph which connects the activated nodes. Formal graph model can be described as follows:

Vertices: For each tuple T in the database, the graph has a corresponding node u_T.

Edges: For each pair of tuples T_1 and T_2 such that there is a foreign key from T_1 to T_2, the graph contains an edge from T_1 to T_2 and a back edge from T_2 to T_1 (this can be extended to handle other types of connections). [Hulgeri, 2001]

2.3 BANKS

Banks [Aditya, 2002] works on graph representation of relational database. An answer to a query is considered to be a rooted directed tree containing a directed path from the root to each keyword node. The root node is called an 'information node' and the tree a 'connection tree'.

2.3.1 Database and query model

Node weights: Each node u in the graph is assigned a weight N(u) which depends upon the prestige of the node. Node weights are inspired by prestige rankings such as PageRank in Google. With this feature, nodes that have multiple pointers to them get a higher prestige. In current implementation the node prestige is set to the in-degree of the node. Higher node weight corresponds to higher prestige.

Edge weights: If u, v are nodes in the database graph and R(u) and R(v) are the relations they belong to. s(R(u), R(v)) denotes the similarity between two relations. And IN v (u) is the in-degree of u contributed by tuples belonging to relation R(v). Between nodes u and v conceptually may have two edges. Weight of forward edge is set to s(R(u), R(v)) and that of reverse edge is set to [s(R(v), R(u)) * IN v (u)]. The actual edge weight is the minimum of the two.

The weight of a backward link generated from a foreign key relationship is directly proportional to the in-degree of the source node (i.e. the referenced node). Since the proximity between the nodes connected by a link is inversely proportional to the link weight, the proximity from a referenced node to its referencing nodes is inversely proportional to the in-degree of the referenced node.

An answer to a query is a rooted directed tree containing at least one node from each Si. Note that the tree may also contain nodes not in any Si and is therefore a ‘Steiner tree’. The relevance score of an answer tree is computed from the relevance scores of its nodes and its edge weights. (The condition that one node from each Si must be present can be relaxed to allow answers containing only some of the given keywords.)

2.3.2 Backward expanding search algorithm

Finding minimum steiner tree is NP-complete problem. Backward expanding search algorithm offers a heuristic solution for incrementally computing query results. The algorithm description is mentioned bellow:

Input: set of keywords t_1, t_2, ……., t_n

Output: Connection trees with highest relevance score containing all keywords in the query.

- For each term t_i and the set of nodes Si relevant to keyword by using disk resident indices on keyword.

- Let S = UiSi Backward search concurrently runs |S| copies of Dijkstra's single source shortest path algorithm for each keyword node n in S, with n as source.

- All copies run concurrently by creating iterator interface.

- Each copy traverses graph in reverse direction. To find common vertex from which forward path exists to at least one node in each set Si. In each Iteration algorithm picks an iterator whose next vertex to be output is at least distance from source vertex of iterator.

- Generate all possible connection trees. They are approximately sorted. Maintain small fixed-size heap of generated connection trees ordered on relevance of the tree.

    - Keep adding newly generated connection trees to heap.

    - When heap is full, output highest relevant node in the heap and add new tree to the heap.

    - When all answers are generated, remaining trees in heap are output in decreasing order of relevance.

Generating all connection trees and then sorting in decreasing relevance order would increase computation costs and increased time to generate initial results. To avoid these overheads, as a heuristic, the technique maintains a small fixed-size heap of generated connection trees. This heuristic does not guarantee that the trees are generated in decreasing order, it is found that it works well even with a reasonably small heap size.

2.3.3 User Feedback

Keyword queries are inherently ambiguous, so a user may need to interact with the system to find required answers. ***Banks*** provides several strategies for refining queries to get required results [Aditya, 2002].

• **Disambiguation of nodes:** A given keyword may match several nodes. ***Banks*** allows users to select which nodes are relevant and re-execute the query with those nodes.

• **Answer patterns:** Suppose a user wishes to find papers by Soumen that refer to papers by Sudarshan, and executes a query sudarshan soumen. This query would return papers written jointly by Sudarshan and Soumen, papers by Sudarshan that refer to papers by Soumen in addition to papers by Soumen that refer to papers by Sudarshan. The ***Banks*** system allows the users to select particular tree patterns as relevant and find only answers that match that pattern. The tree patterns are used to prune the search for answers.

• **Re-scoring:** A feature for the user to express a softer preference, by simply marking some answers as relevant (or more relevant than unmarked answers) is added in ***Banks*** system. The random walk model for answer tree scoring can use this information to prefer or avoid certain paths.

2.4 DBXplorer

Given a set of query keywords, ***DBXplorer*** [Agrawal, 2002] returns all rows (either from single tables, or by joining tables connected by foreign-key joins) such that the each row contains all keywords. Enabling such keyword search requires (a) a preprocessing step called Publish that enables databases for keyword search by building the symbol table and associated structures, and (b) a Search step that gets matching rows from the published databases.

2.4.1 Publish procedure

A database (or a desired part of it) is enabled for keyword search through the following steps. Initially a database is identified, along with the set of tables and columns within the database to be published. Then Symbol Tables are created for supporting keyword searches, which is used at search time to efficiently determine the locations of query keywords in the database (i.e., the tables, columns, rows they occur in). The 3 types of symbol tables based on granularity are:

(1) Column level granularity (Pub-Col), where for every keyword the symbol table maintains the list of all database columns (i.e., list of table. column) that contain it. They are usually much smaller in size hence lead to efficient search if index on column is present.

(2) Cell level granularity (Pub-Cell), where for every keyword the symbol table maintains the list of database cells (i.e., list of table.column,rowid) that contain it. Search is faster in if indices on columns are not available but update cost is much more.

(3) Hybrid Structure which is needed where the granularity is tied to the physical database design: if an index is available for a column, it will be better to publish the column contents with Pub-Col granularity, otherwise with Pub-Cell granularity.

2.4.2 Search procedure

Given a query consisting of a set of keywords, it is answered as follows. Step 1: The symbol table is looked up to identify the tables, and columns/rows of the database that contain the query keywords. Step 2: All potential subsets of tables in the database that, if joined, might contain rows having all keywords, are identified and enumerated. A subset of tables can be joined only if they are connected in the schema, i.e., there is a sub-tree (called a join tree) in the schema graph that contains these tables as nodes (and possibly some intermediate nodes). Step 3: For each enumerated join tree, a SQL statement is constructed (and executed) that joins the tables in the tree and selects those rows that contain all keywords. The final rows are ranked and presented to the user.

Some key parts of this procedure are expanded below:

Enumerating join trees: Let matched tables be the set of database tables that contain at least one of the query keywords. If the schema graph G is viewed as an undirected graph, this step enumerates join trees, i.e., sub-trees of G such that: (a) the leaves belong to matched tables and (b) together, the leaves contain all keywords of the query. Thus, if the tables that occur in a join tree are joined, the resulting relation will contain all potential rows having all keywords specified in the query. This important step filters out a large number of spurious join scenarios from being passed on to the subsequent step of the search. If G is not a tree (i.e., if it contains cycles), the join tree enumeration involves bi-connected component decomposition of G, followed by the enumeration of join trees on a possibly cyclic schema graph. This feature is currently not supported by ***DBXplorer.***

Searching for rows: The input to this final search step is the enumerated join trees. Each join tree is then mapped to a single SQL statement that joins the tables as specified in the tree, and selects those rows that contain all keywords. In fact, this is the only stage of the search where the database tables are accessed.

Outputting results: The retrieved rows are ranked before being output. The approach here is to rank the rows by the number of joins involved (ties broken arbitrarily); the reasoning being that joins involving many tables are less relevant.

2.4.3 Execution efficiency

It depends on several factors e.g. availability of column indexes for the Pub-Col based approach. There may be commonalities among the generated SQL statements for a given keyword search query, with potential applications of multi-query optimization for further efficiency. Since enumeration algorithm generates join trees in order of increasing size (due to breadth first enumeration), the join tree enumeration step can be pipelined and thus followed immediately by the SQL generation corresponding to the join tree which further reduces response time.

2.5 DISCOVER

Discover [Hristidis, 2002] outputs query results by generating intermediate SQL queries and evaluating them efficiently by generating nearly optimal plan. It introduces a concept of candidate networks i.e. join expressions on foreign to primary key relationships of relations or tuple sets.

Minimal Total Joining Network of Tuples (MTJNT) of keywords k_1, k_2, ..., k_m is the set of all possible joining networks of tuples that are both:

• **Total:** every keyword is contained in at least one tuple of the joining network.

• **Minimal:** it is not possible to remove any tuple from the joining network and still have a total joining network of tuples.

2.5.1 Architecture

This technique proceeds in two stages:

- Candidate Network Generator generates all candidate networks of relations, that is, join expressions that generate the joining networks of tuples.

- Plan Generator builds plans for the efficient evaluation of the set of candidate networks, exploiting the opportunities to reuse common sub expressions of the candidate networks.

Operation of *Discover*:

Figure (2.1) shows the basic stepwise operation of ***Discover***. It is described in brief below:

Step 1: Master Index, this step builds full-text indices on single attributes using existing database system utilities.

Step 2: Tuple set post processor, takes as input basic tuple sets and produces tuple sets containing all keywords of query and no other keyword.

Step 3: Candidate Network Generator, given a set of keywords k_1, k_2, ...k_m a candidate network C is a joining network of tuple sets such that it has MTJNT (Minimal Total Joining Networks of Tuples) M belongs to C and no tuple t belongs to M that maps to a free tuple set F belonging to C and containing any keywords.

A joining network of tuples j is not minimal if :

- It has a tuple with no keywords as a leaf. ***Discover*** eliminates this condition for joining networks of tuple sets by not allowing free tuple sets as leaves.

- It contains sample tuple twice. Here, a candidate network does not contain a subtree of the form $R^K \rightarrow S^L \leftarrow R^M$, where R and S are relations and the schema graph has an edge $R \rightarrow S$.

Thus it gives minimal joining network of tuple sets.

Step 4: Plan Generator, in this step each rule will produce intermediate result or final result. The efficiency is improved through reuse of common join sub expression.

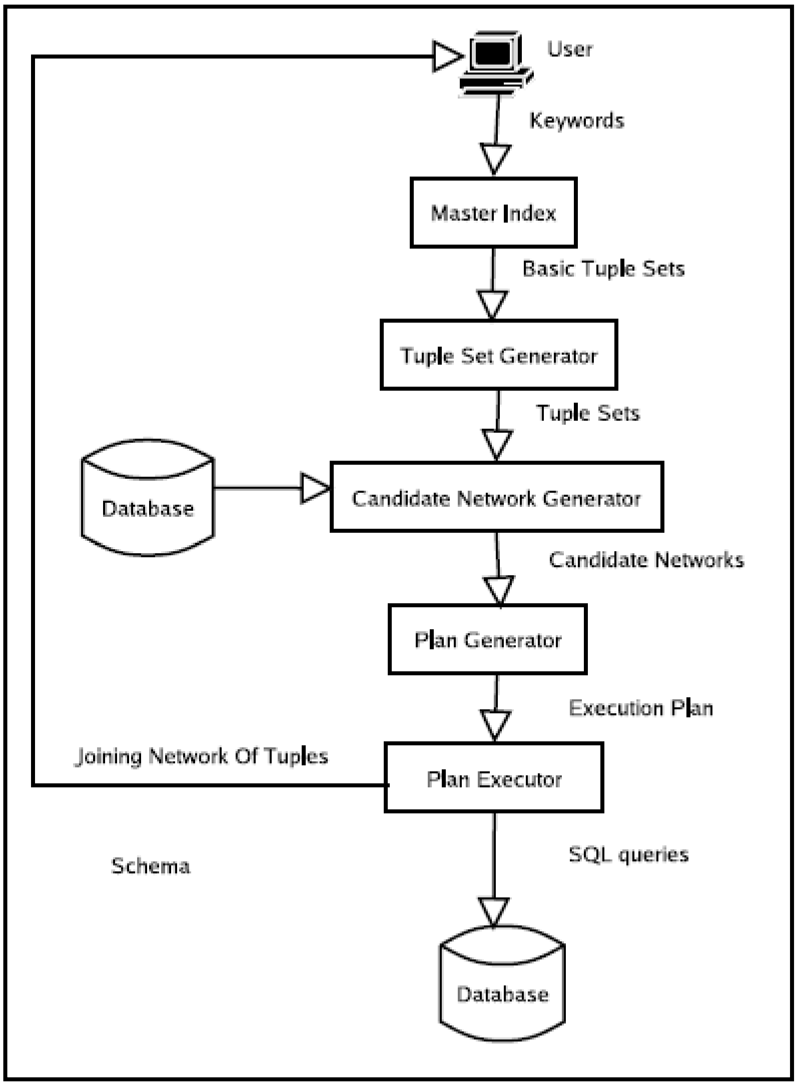


Figure 2.1: Discover Architecture.

Given a set of candidate networks, find the intermediate results that should be built, so that the overall cost of building these results and evaluating the candidate networks is minimum is NP-complete on the size of the candidate networks with respect to the theoretical cost model. ***Discover*** uses greedy algorithm and quality of the plans produced by the algorithm are very close to the quality of the optimal plans.

Step 5: Plan Execution Module, this module prepares SQL statements from execution plan and executes them over underlying database.

2.6 Efficient IR style keyword search

Applications in which plain text coexists with structured data are pervasive. Commercial relational database management systems (RDBMSs) generally provide querying capabilities for text attributes that incorporate state-of-the-art information retrieval (IR) relevance ranking strategies, but this search functionality requires that queries specify the exact column or columns against which a given list of keywords is to be matched. The technique of [Hristidis, 2003] makes use of these available features of underlying RDBMS and overcomes the above mentioned problem, due to which user with absolutely no information about database schema can query the database easily.

2.6.1 System Architecture

The system architecture is shown in Figure (2.2). It consists of following main blocks:

• **IR Engine:** This module exploits IR-style text indexing functionality of Modern RDBMSs to identify all tuples that have a non-zero score functionality for a given query. The IR Engine relies on the IR Index, which is an inverted index that associates each keyword that appears in the database with a list of occurrences of the keyword, each occurrence of a keyword is recorded as a tuple attribute pair.

• **Score Assignment:** Score assigned to joining tree of tuples depends on: (1) Single attribute IR-style relevance score score(a_i, Q) for each textual attribute $a_i \in T$ and query Q as determined by an IR engine at the RDBMS. This is tf-idf score of the attribute. (2) A function combine which combines the single attribute scores into a final score of T. It is equal to the ratio of summation of individual attribute scores to size of T. IR engine takes query Q as input and extracts for each relation R tuple set $R^Q = \{t \in R|Score(t, Q) > 0\}$.

• **Candidate Network Generator:** Candidate network S is a join network that involves tuple sets plus additional (base) relations. Base relation R that appears in a CN is referred as a free tuple set and denote it as $R^{\{\}}$. Here only a single tuple set R^Q is created for each relation R as specified above. For queries with AND semantics, post processing step checks that it only returns tuple trees containing all query keywords.

They consider S to be a Candidate network and hence part of output if it satisfies following properties: (1) No. of non-free tuple sets in S does not exceed the number of query keywords m. (2) No leaf tuple sets of S are free. (3) S does not contain a construct of the form R → S ← R. The size of the Candidate Network is its number of tuple sets.

• **Execution Engine:** It takes as input CNs together with non-free tuple sets. It contacts RDBMSs query execution engine repeatedly to identify top-k query results.

There are many algorithms for improving efficiency of this step rather Naive algorithm: (1) Sparse Algorithm: This approach computes a bound MPSI i.e. Maximum Possible Score of a tuple tree derived from a Candidate Network C; If MPSI does not exceed actual score of k already reduced tuple trees then CN C_i can be safely removed from further consideration. (2) Single Pipelined Algorithm: This algorithm maintains an effective estimate of MPFS (Maximum Possible Future Score) that an unseen result can achieve given the information already gathered by algorithm. Hence this algorithm can start producing results before examining the entire tuple sets as if the score of any result in the queue has score greater than MPFS, then it is safe to output it as it will be one of top-k results. (3) Global Pipelined Algorithm: It builds on the Single Pipelined algorithm to efficiently answer a top-k keyword query over multiple CNs. The algorithm receives as input a set of candidate networks, together with their associated non-free tuple sets, and produces as output a stream of joining trees of tuples ranked by their overall score for the query. The key idea of the algorithm is the following. All CNs of the keyword query are evaluated concurrently following an adaptation of a priority preemptive, round robin protocol, where the execution of each CN corresponds to a process. Each CN is evaluated using a modification of the Single Pipelined algorithm, with the priority of a process being the MPFS value of its associated CN.

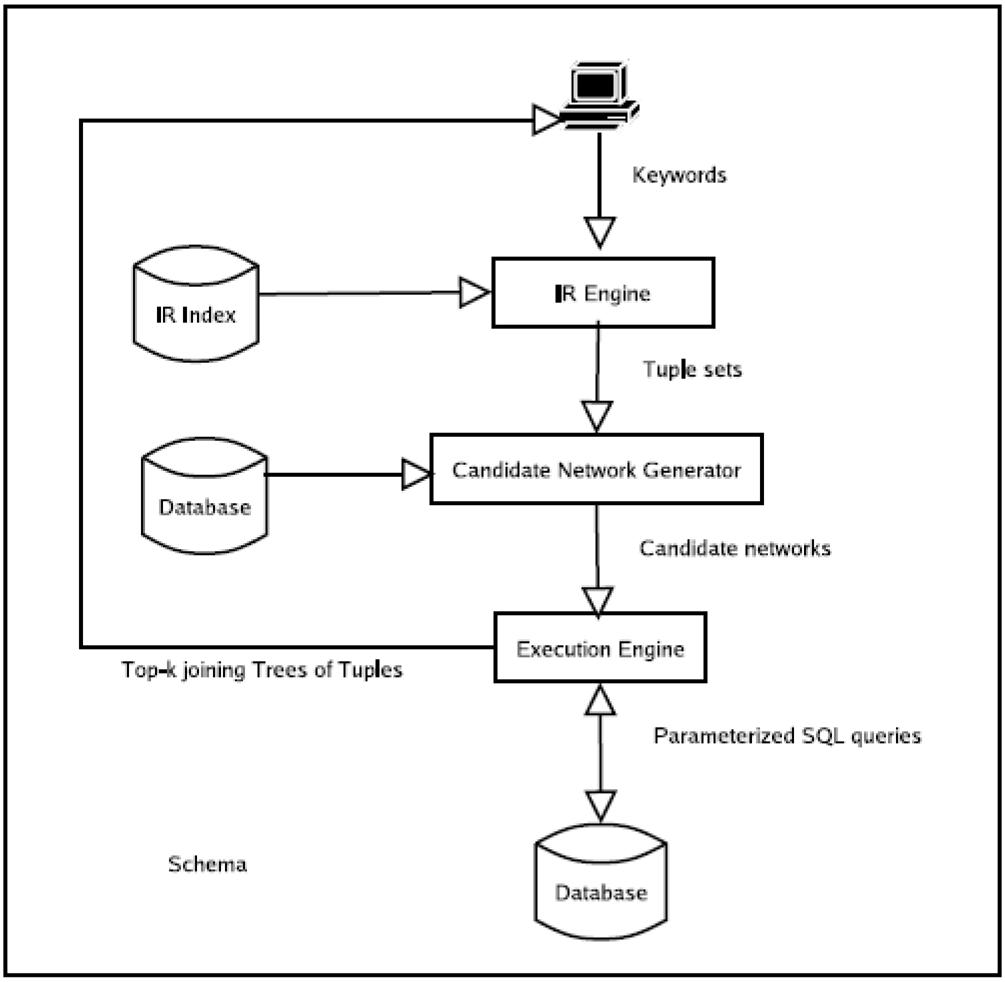


Figure 2.2: IR style keyword search architecture.

2.7 ObjectRank

ObjectRank [Balmin, 2004] works on database modeled as labeled graph. Authority originates at nodes containing keywords and flow from one object to another according to semantic connection between them. Each node is ranked according to its authority with respect to particular keywords. This is different from other techniques, in a way that most of the other techniques assign a ranks to particular node based on original database and it is not influenced by query keywords. Here keyword specific ObjectRank is also taken into consideration while calculating rank of node.

They can adjust: (1) weight of global importance. (2) Weight of each keyword of query. (3) Importance of result actually containing keyword vs. being referenced. (4) Volume of authority flow via each type of semantic connection.

Authority Transfer Schema Graph and Authority Transfer Data Graph are constructed from Schema graph and Database graph respectively, which control the authority flow from one node to another.

2.7.1 ObjectRank Architecture

Architecture is divided into two stages.

Preprocessing stage:

1. ObjectRank Execution module takes as input the database to be indexed, the set of all keywords that will be indexed, and a set of parameters. It creates an ObjectRank index which is an inverted index indexed by keywords of query. The score of a node v with respect to a keyword query w is a combination of the global ObjectRank rG(v) of v and the keyword-specific ObjectRank rw(v). The following combining function is currently being used: $r^{w,G}(v) = r^{w}(v)(r^{G}(v))^{g}$.

2. For each keyword w, it stores a list of < id(u); $r^{w}(u)$ > pairs for each object u with $r^{w}(u) \geq$ threshold. The pairs are sorted by descending $r^{w}(u)$ to facilitate an efficient querying method as described below. The ObjectRank Index has been implemented as an index-based table, where the lists are stored in a CLOB attribute. A hash-index is built on top of each list to allow for random access, which is required by the Query module.

Search stage:

1. Query module outputs top-k objects according to combining function using threshold algorithm. The Threshold Algorithm is applicable since both combining functions (for AND and OR semantics) are monotone and random access is possible on the stored lists. Output of this algorithm is IDs of top k results.

2. Database Access module inputs the result ids and queries the database to get the suitable information to present the objects to the user. This information is stored into an id-indexed table that contains a CLOB attribute value for each object id.

2.8 Comparisons of different techniques

Banks considers node weights, forward and backward edge weights and similarity between relations to find relevance of the answer. While ranking in ***DBXplorer*** and ***Discover*** is only dependent on size of joining network.

Banks assumes that graph fits in memory. ***DBXplorer*** and ***Discover*** do not need such in memory data structures, hence the maximum size of database they can process is not bounded by memory size.

The technique of [Hristidis, 2003] is specifically designed for applications in which plain text coexists with structured data e.g. comments or remarks column in a relational database. It fully exploits single-attribute relevance-ranking results if the underlying RDBMS has text indexing capabilities. This technique produce top-k results in a pipelined fashion, in the sense that execution can resume to compute the "ext-k" matches if the user so desires. This second feature is different from all other techniques.

Discover and ***DBXplorer*** create a separate tuple set for each combination of keywords in query Q and each relation. This generally leads to a number of CNs that is exponential in the query size, which makes query execution prohibitively expensive for queries of more than a very small number of keywords. Technique of [Hristidis, 2003] creates only a single tuple set R^Q for each relation R.

For queries with AND semantics, a post-processing step checks that only tuple trees containing all query keywords are returned. This characteristic of the system results in significantly faster executions, which in turn allows handling larger queries and also considering larger CNs.

ObjectRank initially calculates global rank of an element in the document and then calculates keyword specific rank at runtime. It recomputed an inverted index where for each keyword there is a sorted list of the nodes with non-trivial ObjectRank for each keyword. During runtime the Threshold algorithm is employed to efficiently combine the lists. It performs well compared to global ranking algorithm like Google's PageRank and simple algorithms which do not consider authority transfer between nodes.

Chapter Three: Background

This chapter gives the basic concepts and definitions necessary to best understanding of this thesis. As the subject is vast, we limit the information to what is useful and necessary for this work

3.1 Relational Database

A relational database was developed by Edgar Codd in 1969, it is a structured collection of information that is related to a particular subject or purpose, such as a library database or a human resources database. Within the database, you organize the data into storage containers called tables. Tables are made up of columns and rows. Columns represent individual fields in a table. Rows represent records of data in a table.

Each field in the table contains one piece of information. In a book table, for example, one column contains the book title, another contains the ISBN, and numbers of pages are all stored in their own columns. Each record represents one set of related information. For example, the book table might store information about one book per row.

In a database, you can organize data in multiple tables. For example, if you manage a database for the library department, you might have one table that lists all the books information and another table that lists all the authors.

The tables of the database are linked by the primary-foreign key relationships, where a primary key - foreign key relationship defines a one-to-many relationship between two tables in a relational database. The term foreign key is defined as a column or a set of columns in one table that references the primary key columns in another table, and the primary key is defined as a column (or set of columns) where each value is unique and identifies a single row of the table.

3.2 Extracting data from database

There are different methods available today to extract information from relational database, choosing the method of retrieving data depends on the kind of user (naïve or expert user) and his needs. We can extract data from database using one of the following methods: SQL, QBE, Subject search or Keyword search. Each one of them has its own characteristic and limitations. To demonstrate how we can formulate the query for each method, we will generate the queries over the Library database which has the following schema (see Figure (3.1)).

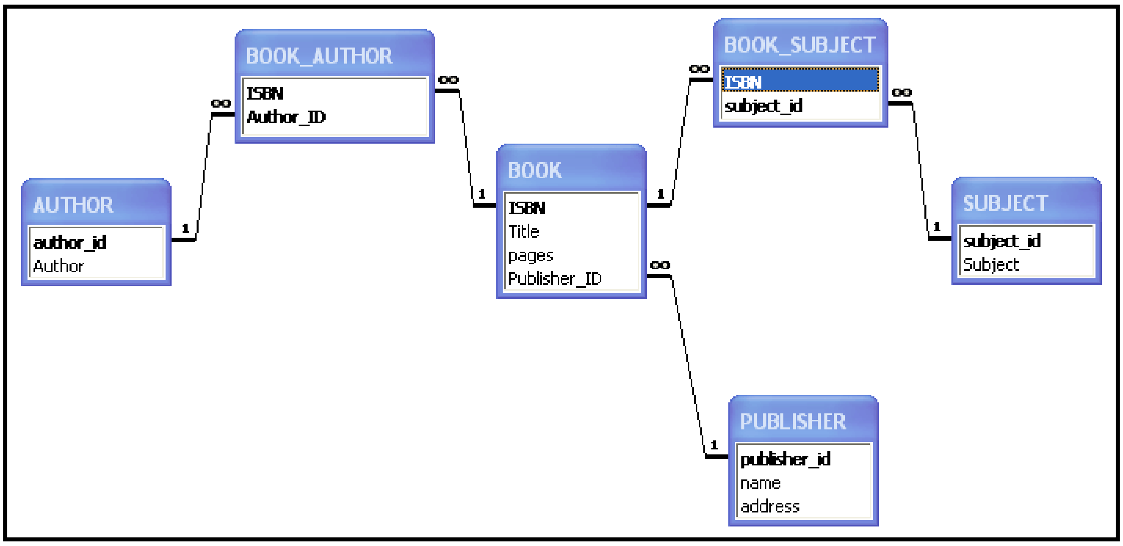


Figure 3.1: ER- Diagram of the Library database.

3.2.1 SQL

SQL (Structured Query Language) is a language that lets you communicate with databases. For example, you can use SQL to retrieve data from a database, add data to a database, delete or update records in a database, change columns in multiple rows, add columns to tables and add and delete tables. SQL was one of the first languages for Edgar F. Codd's relational model in his influential 1970 paper [E.F. Codd, 1970].

The SELECT statement is the most frequently used of all SQL commands after a database's establishment. The SELECT statement allows you to view data that is stored in the database. It is not a standalone statement, which means that one or more additional clauses (elements) are required for a syntactically correct query. In addition to the required clauses, there are optional clauses that increase the overall functionality of the SELECT statement.

There are four keywords, or clauses, that are valuable parts of a SELECT statement. These keywords are as follows: SELECT, FROM, WHERE and ORDER BY, for more illustration see [http://www.w3schools.com/sql/default.asp].

SQL query

For example, if the user wants to retrieve information about the books in the domain of statistics which is written by Rusell, the user must type the following query:

```
select book.isbn, book.title, author.author
 from book inner join (author inner join book_author on author.author_id =
book_author.author_id) on book.isbn = book_author.isbn
 where (((book.title) like ‘*statistic*’) and ((author.author) like ‘*Rusell*’));
```

Contents of a Query

Most queries require at least the following conditions to be stated. First, which table or tables is the data coming from? If from two or more tables, what is the link between? Next, define the selection criteria, which is the matching condition or filter. Lastly, define which fields in the tables are to be displayed or printed in the result.

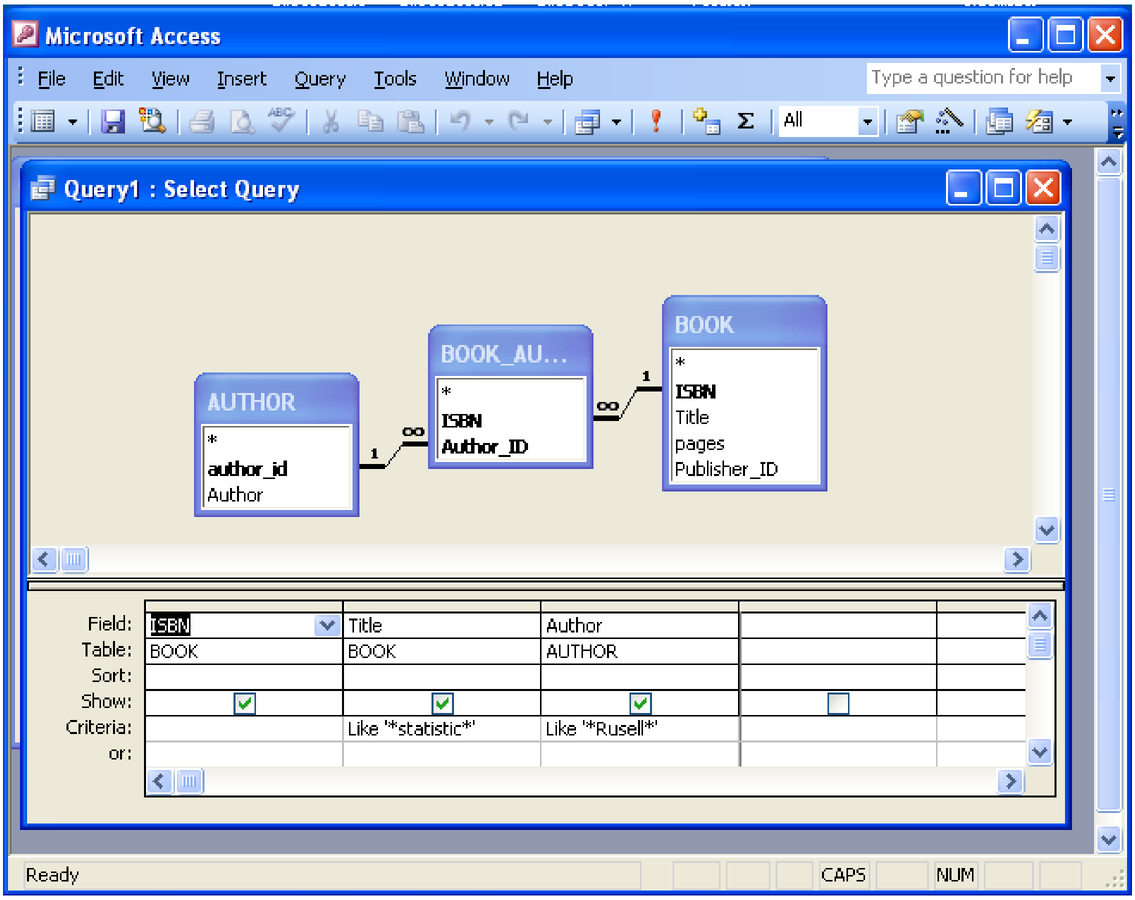
3.2.2 QBE

Query by Example (QBE) is a database query language for relational databases. It was devised by Moshé M. Zloof at IBM Research during the mid 1970s, in parallel to the development of SQL. It is the first graphical query language, using visual tables where the user would enter commands, example elements and conditions.

The main purpose of QBE is to make it easier to run database queries and to avoid the frustrations of SQL errors. QBE converts the user's input into a database SQL query.

QBE query

For example, if the user wants to retrieve information about the books in the domain of statistics which is written by Rusell, the user will use a graphical user interface as shown in Figure (3.2).



Field:	ISBN	Title	Author	
Table:	BOOK	BOOK	AUTHOR	
Sort:				
Show:	☑	☑	☑	□
Criteria:		Like "*statistic*"	Like "*Rusell*"	
or:				

Figure 3.2: QBE interface for Ms Access.

This interface, allows the user to perform powerful searches without having to learn SQL, the user just fills in blanks and select items to define the query he/she wants to perform.

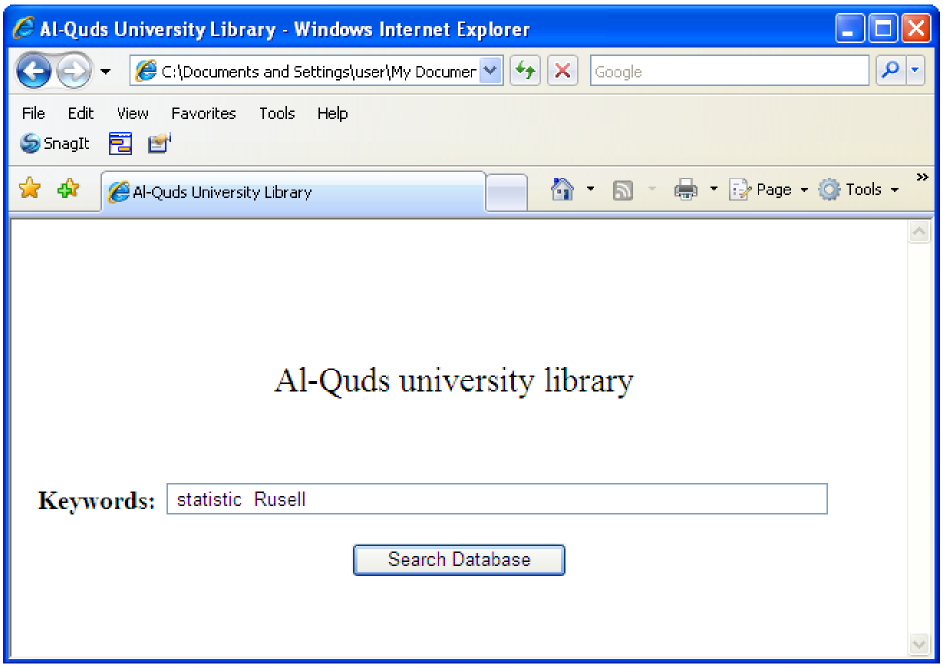
3.2.3 Keyword Search

Keyword searches are similar to Internet searches with Google in that the database will look for the words you use wherever they may be on a page. Regardless of whether the word is in a title, author name or place of publication, the page will be returned as a result.

i.e. When the user do a keyword search in a library catalog or a database, he can type in words that describe his research topic in any order and retrieve records containing those search terms. A major disadvantage of a keyword search is that it does not take into account the meaning of the words used as search terms, so if a term has more than one meaning (such as "mouse" - computer hardware or rodent?), irrelevant records may be retrieved.

Keyword search query

For example, if the user wants to retrieve information about the books in the domain of statistics which is written by Rusell, the user will use a graphical user interface as shown in Figure (3.3).


Al-Quds University Library - Windows Internet Explorer
C:\Documents and Settings\user\My Documer
Google
File Edit View Favorites Tools Help
SnagIt
Al-Quds University Library
Page
Tools
Search Database

Al-Quds university library

Keywords: statistic Rusell

Figure 3.3: Keyword search interface.

The user can use this page to search the full catalog for any keyword. Whatever word or words he type into the search box, a result will be returned if a match is found anywhere in the record, including the author, title and subject fields.

In general, the user types his choice of keywords into the text box. Then click the search button to begin his search.

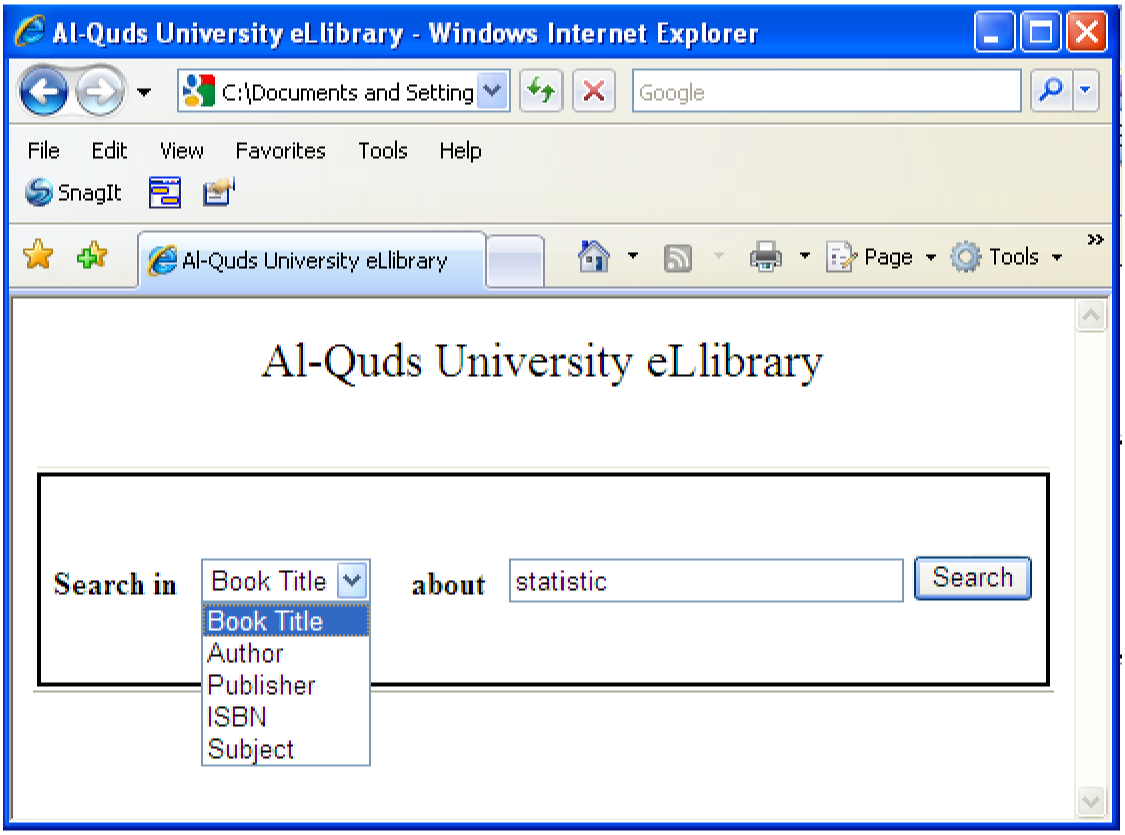
3.2.4 Subject Search

Subject searches, only return results in which the term being used appears in the subject heading.

i.e. When the user do a subject search in a library catalog or database, only the subject headings (subject fields) are searched for words that match his search terms. In library catalogs and databases, items are assigned subject headings as access points, to assist users in locating the content. Using subject headings ensures that all items about the same topic have consistent subject headings and so they can all be accessed with one search term. This saves the user time.

Subject search query

For example, if the user wants to retrieve information about the books in the domain of statistics which is written by Rusell, the user will use a graphical user interface as shown in Figure (3.4).


Al-Quds University eLlibrary - Windows Internet Explorer
C:\Documents and Setting
Google
File
Edit
View
Favorites
Tools
Help
SnagIt
Al-Quds University eLlibrary
Page
Tools

Al-Quds University eLlibrary

Search in: Book Title

about: statistic

Figure 3.4: Subject search interface.

The user choose the Book Title from the drop down list, enter statistic keyword in the search box and then click the search button to begin his search.

In general, the user can use this page to limit the search to the stock of one particular branch by choosing the branch from the drop-down list.

3.2.5 SQL Vs. QBE

QBE is a language for querying like SQL, for retrieving, creating and modifying relational data. It is different from SQL in having a graphical user interface that allows users to write queries by creating example tables on screen. A user needs minimal information to get started and the whole language contains relatively few concepts. QBE is especially suited for queries that are not too complex and can be expressed in terms of a few tables. Table (3.1) summarizes the differences between SQL and QBE.

Table 3.1: Differences between SQL and QBE.

QBE	**SQL**
Graphical user interface	Command line
Very convenient for simple queries	More suitable for complex queries
Need minimal information to get	Must understand a lot of concepts

3.2.6 Keyword search vs. SQL

SQL and keyword search are two widely different techniques in discovery information, in the first one the user needed to understand the rules of the language to communicate with databases, while the second one the user only type a list of keywords in the text box that related with his search topic to communicate with databases, Table (3.2) shows some key differences between them.

Table 3.2: Differences between keyword search and SQL.

Keyword search	**SQL**
Display a big amount of results, some are relevance, other are not.	SQL returns exactly what the user needs
Ranking the results according relevancy	Not ranking the results
Limited for retrieving data from the database	Used to define, access, and manipulate data
Keywords query can formulate by any kind of user	SQL queries can formulate only by expert

3.2.7 Keyword vs. Subject Search

Keyword Search and Subject search are common search techniques that the ordinary user can apply to any database. These will enable the user to retrieve relevant information from the thousands of records in a database. There are several differences between subject searches and keyword searches. Table (3.3) shows some key differences between them.

Table 3.3: Differences between keyword search and subject search.

Keyword search	**Subject search**
Search for: Records that have the search term anywhere within them.	**Search for:** Records that have the search term in the subject headings part of that record.
Volume: Depending on the terms you use, searches may retrieve no results or thousands. Searches with general terms often return many results.	**Volume:** Varies widely. Some searches will retrieve hundreds of results, but, if you choose a nonexistent subject term, you will get none.
Relevance: Varies. Results may be completely unrelated to your topic. For example, a search for 'Philadelphia' returns records for every book published by the University of Pennsylvania Press (located in 'Philadelphia') regardless of whether the work is about Philadelphia.	**Relevance:** High as long as you identify the correct subject for your topic.
Flexibility: High: Terms can be combined in complex ways to design effective searches.	**Flexibility:** The flexibility of your search is limited by the manner in which subjects are structured in the database that you are searching.

The most obvious is keyword searches are broader searches than subject searches.

3.3 Stemming

Stemming algorithm attempt to reduce a word to its stem or root form. Thus, the keyword terms of a query or records in the keyword search engine over relational database are represented by stems rather than by the original words. This not only means that different variants of a term can be conflated to a single representative form – it also reduces the dictionary size, that is, the number of distinct terms needed for representing a set of records. A smaller dictionary size results in a saving of storage space and processing time. For example, if the user enters "viewer" as the query, the search engine reduces the word to its root "view" and returns all records containing the root - like records containing view, viewer, viewing, preview, review etc. [Lovins, 1968] is the first paper on the subject was published.

3.4 Stop Words

A stop word is a commonly used word (such as "the") that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query. When building the index, most engines are programmed to remove certain words from any index entry. The list of words that are not to be added is called a stop list. Stop words are deemed irrelevant for searching purposes because they occur frequently in the language for which the indexing engine has been tuned. In order to save both space and time, these words are dropped at indexing time and then ignored at search time. Table (3.4) shows a comprehensive list of words ignored by our Search Engine ***Ssearch***.

Table 3.4-a: Stopwords [www.wenconfs.com].

A	Able	About	Above	Abroad	According	accordingly
Across	Actually	Adj	After	Afterwards	Again	Against
Ago	Ahead	ain't	All	Allow	Allows	Almost
Alone	Along	Alongside	Already	Also	Although	Always
Am	Amid	Amidst	Among	Amongst	An	And
Another	Any	Anybody	Anyhow	Anyone	Anything	Anyway
Anyways	Anywhere	Apart	Appear	Appreciate	appropriate	Are
Aren't	Around	As	a's	Aside	Ask	Asking
associated	At	Available	Away	Awfully	B	Back
Backward	Backwards	Be	Became	Because	become	becomes

Table 3.4-b: Stopwords.

Becoming	Been	Before	Beforehand	Begin	Behind	Being
Believe	Below	Beside	Besides	Best	Better	Between
Beyond	Both	Brief	But	By	C	Came
Can	Cannot	Cant	can't	Caption	Cause	Causes
Certain	Certainly	Changes	Clearly	c'mon	Co	co.
Com	Come	Comes	Concerning	consequently	consider	considering
Contain	Containing	Contains	corresponding	Could	couldn't	Course
c's	Currently	D	Dare	daren't	definitely	described
Despite	Did	Didn't	Different	Directly	Do	Does
doesn't	Doing	Done	don't	Down	downwards	During
E	Each	Edu	Eg	Eight	Eighty	Either
Else	Elsewhere	End	Ending	Enough	Entirely	especially
Et	Etc	Even	Ever	Evermore	Every	everybody
Everyone	Everything	Everywhere	Ex	Exactly	example	Except
F	Fairly	Far	Farther	Few	Fewer	Fifth
First	Five	Followed	Following	Follows	For	Forever
Former	Formerly	Forth	Forward	Found	Four	From
Further	Furthermore	G	Get	Gets	Getting	Given
Gives	Go	Goes	Going	Gone	Got	Gotten
Greetings	H	Had	hadn't	Half	happens	Hardly
Has	Hasn't	Have	haven't	Having	He	he'd
he'll	Hello	Help	Hence	Her	Here	hereafter
Hereby	Herein	Here's	Hereupon	Hers	Herself	he's
Hi	Him	Himself	His	Hither	hopefully	How
Howbeit	However	Hundred	I	i'd	Ie	If
Ignored	i'll	i'm	Immediate	In	inasmuch	Inc
inc.	Indeed	Indicate	Indicated	Indicates	Inner	Inside
Insofar	Instead	Into	Inward	Is	isn't	It
it'd	it'll	Its	it's	Itself	i've	J
Just	K	Keep	Keeps	Kept	Know	Known
Knows	L	Last	Lately	Later	Latter	Latterly
Least	Less	Lest	Let	let's	Like	Liked
Likely	Likewise	Little	Look	Looking	Looks	Low
Lower	Ltd	M	Made	Mainly	Make	Makes
Many	May	Maybe	mayn't	Me	Mean	meantime
meanwhile	Merely	Might	mightn't	Mine	Minus	Miss
More	Moreover	Most	Mostly	Mr	Mrs	Much
Must	mustn't	My	Myself	N	Name	Namely
Nd	Near	Nearly	Necessary	Need	needn't	Needs
Neither	Never	Neverf	Neverless	nevertheless	New	Next
Nine	Ninety	No	Nobody	Non	None	nonetheless
Noone	No-one	Nor	Normally	Not	nothing	notwithstanding
Novel	Now	Nowhere	O	Obviously	Of	Off
Often	Oh	Ok	Okay	Old	On	Once
One	Ones	One's	Only	Onto	opposite	Or
Other	Others	Otherwise	Ought	Oughtn't	Our	Ours
Ourselves	Out	Outside	Over	Overall	Own	P
Particular	Particularly	Past	Per	Perhaps	placed	Please
Plus	Possible	Presumably	Probably	Provided	provides	Q
Que	Quite	Qv	R	Rather	Rd	Re
Really	Reasonably	Recent	Recently	Regarding	regardless	Regards

Table 3.4-c: Stopwords.

Relatively	Respectively	Right	Round	S	Said	Same
Saw	Say	Saying	Says	Second	secondly	See
Seeing	Seem	Seemed	Seeming	Seems	Seen	Self
Selves	Sensible	Sent	Serious	Seriously	Seven	Several
Shall	Shan't	She	she'd	she'll	she's	Should
shouldn't	Since	Six	So	Some	somebody	someday
Somehow	Someone	Something	Sometime	sometimes	somewhat	somewhere
Soon	Sorry	Specified	Specify	Specifying	Still	Sub
Such	Sup	Sure	T	Take	Taken	Taking
Tell	Tends	Th	Than	Thank	thanks	Thanx
That	that'll	that’s	that's	that've	The	Their
Theirs	Them	Themselves	Then	Thence	There	thereafter
Thereby	There'd	Therefore	Therein	there'll	there're	Theres
There's	Thereupon	There've	These	They	they'd	they'll
They're	They've	Thing	Things	Think	Third	Thirty
This	Thorough	Thoroughly	Those	Though	Three	Through
throughout	Thru	Thus	Till	To	together	Too
Took	Toward	Towards	Tried	Tries	Truly	Try
Trying	t's	Twice	Two	U	Un	Under
underneath	Undoing	Unfortunately	Unless	Unlike	unlikely	Until
Unto	Up	Upon	Upwards	Us	Use	Used
Useful	Uses	Using	Usually	V	Value	Various
Versus	Very	Via	Viz	Vs	W	Want
Wants	Was	wasn't	Way	We	we'd	Welcome
Well	we'll	Went	Were	we're	weren't	we've
What	Whatever	What'll	what's	what've	When	Whence
Whenever	Where	Whereafter	Whereas	Whereby	wherein	where's
whereupon	Wherever	Whether	Which	Whichever	While	Whilst
Whither	Who	who'd	Whoever	Whole	Who'll	Whom
whomever	who's	Whose	Why	Will	Willing	Wish
With	Within	Without	Wonder	won't	Would	wouldn't
X	Y	Yes	Yet	You	You'd	you'll
Your	you're	Yours	Yourself	Yourselves	You've	Z
Zero						

3.5 Graph

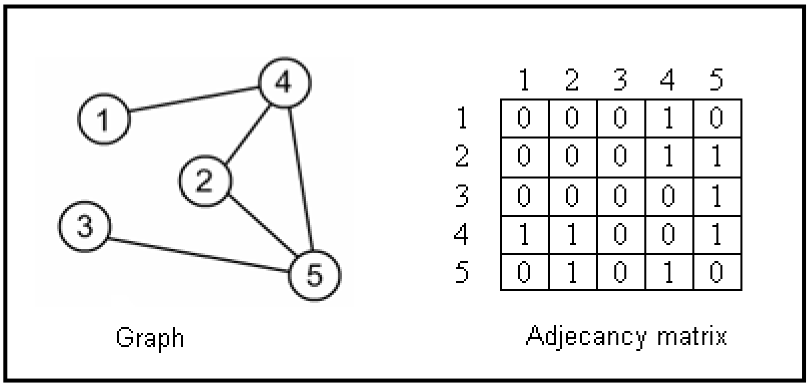
A graph is a collection of vertices or nodes and a collection of edges that connect pairs of vertices. A graph may be undirected, meaning that there is no distinction between the two vertices associated with each edge, or its edges may be directed from one vertex to another. We will discuss undirected graph, which will be used in our work.[Base, 2000]

3.5.1 Undirected graphs representation

There are two ways to represent a graph inside the computer, by using adjacency matrix or by using adjacency list.

3.5.1.1 Adjacency matrix

Each cell a_{ij} of an adjacency matrix contains 1, if there is an edge between i-th and j-th vertices, and 0 otherwise. Figure (3.5) shows an example for undirected graph representation using adjacency m*a*trix.



	1	2	3	4	5
1	0	0	0	1	0
2	0	0	0	1	1
3	0	0	0	0	1
4	1	1	0	0	1
5	0	1	0	1	0

Figure 3.5: Example for undirected graph representation using adjacency matrix

Advantages:

Adjacency matrix is very convenient to work with. Add or remove an edge can be done in O(1) time, the same time is required to check, if there is an edge between two vertices.

Disadvantages:

Adjacency matrix consumes huge amount of memory for storing big graphs. All graphs can be divided into two categories, sparse and dense graphs. Sparse ones contain not much edges (number of edges is much less than the square of number of vertices, $|E| << |V|^2$). On the other hand, dense graphs contain number of edges comparable with square of number of vertices. Adjacency matrix is optimal for dense graphs, but for sparse ones it is superfluous.

Next drawback of the adjacency matrix is that in many algorithms you need to know the edges adjacent to the current vertex. To draw out such an information from the adjacency matrix you have to scan over the corresponding row, which results in O(|V|) complexity.

To sum up, adjacency matrix is a good solution for dense graphs, which implies having constant number of vertices

3.5.1.2 Adjacency list

This kind of the graph representation requires less amount of memory. For every vertex adjacency list stores a list of vertices, which are adjacent to current one. See an example in Figure (3.6).

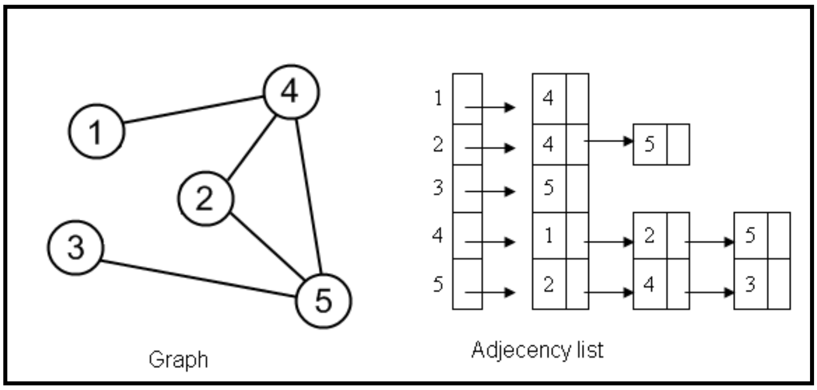


Figure 3.6: Example for undirected graph representation using adjacency matrix.

Advantages:

Adjacent list allows us to store graph in more compact form, than adjacency matrix, but the difference decreasing as a graph becomes denser. Next advantage is that adjacent list allows to get the list of adjacent vertices in O(1) time, which is a big advantage for some algorithms.

Disadvantages:

Adding/removing an edge to/from adjacent list is not so easy as for adjacency matrix. It requires, on the average, O(|E| / |V|) time, which may result in cubical complexity for dense graphs to add all edges.

Check, if there is an edge between two vertices is the next drawback, which can be done in O(|E| / |V|) when list of adjacent vertices is unordered or O(log2(|E| / |V|)) when it is sorted. This operation stays quite cheap.

To sum up, adjacency list is a good solution for sparse graphs.

3.6 Dikjestra's algorithm

Dijkstra's algorithm, conceived by Dutch computer scientist Edsger Dijkstra in 1959, which is a graph search algorithm that solves the single-source shortest path problem for a graph with nonnegative edge path costs, producing a shortest path tree [Zhan, 1998]. This algorithm is often used in routing.

For a given source vertex (node) in the graph, the algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex. It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined. Figure (3.7) demonstrates Dijkstra's algorithm.

In the following algorithm, u := Extract_Min(Q) searches for the vertex *u* in the vertex set *Q* that has the least *d*[*u*] value. That vertex is removed from the set *Q* and returned to the user.

```
 1  function Dijkstra(G, w, s)
 2    for each vertex v in V[G]                    // Initializations
 3          d[v] := infinity
 4          previous[v] := undefined
 5    d[s] := 0
 6    S := empty set
 7    Q := set of all vertices
 8    while Q is not an empty set                  // The algorithm itself
 9          u := Extract_Min(Q)
10          S := S union {u}
11          for each edge (u,v) outgoing from u
12                if d[v] > d[u] + w(u,v)          // Relax (u,v)
13                      d[v] := d[u] + w(u,v)
14                      previous[v] := u
```

If we are only interested in a shortest path between vertices *s* and *t*, we can terminate the search at line 9 if *u = t.*

Now we can read the shortest path from *s* to *t* by iteration:

```
S := empty sequence
u := t
while defined u
      insert u to the beginning of S
      u := previous[u]
```

Now sequence *S* is the list of vertices on the shortest path from *s* to *t.*

Figure 3.7: Dijkstra's Algorithm.

3.7 Quicksort algorithm

Quicksort is a well-known sorting algorithm developed by [Hoare, 1962] that, on average, makes O(nlogn) comparisons to sort n items. However, in the worst case, it makes O(n2) comparisons. Typically, Quicksort is one of the fastest and simplest sorting algorithms. It works recursively by a divide-and-conquer strategy. Figure (3.8) demonstrates the quicksort algorithm.

Input: sequence $a_0, ..., a_{n-1}$ with n elements

Output: permutation of the sequence such that all elements $a_0, ..., a_j$ are less than or equal to all elements $a_i, ..., a_{n-1}$ $(i > j)$

Method:

```
choose the element in the middle of the sequence as comparison element $x$
let $i = 0$ and $j = n-1$
while $i \leq j$
    1.  search for the first element $a_i$ which is greater than or equal to $x$
        search for the last element $a_j$ which is less than or equal to $x$
        if $i \leq j$
            1.  exchange $a_i$ and $a_j$
                let $i = i+1$ and $j = j-1$
```

After partitioning the sequence, quicksort treats the two parts recursively by the same procedure. The recursion ends whenever a part consists of one element only.

Figure 3.8: Quicksort Algorithm.

Chapter Four: Discover Model

4.1 Introduction

One key problem in applying keyword search techniques to databases is that information related to a single answer to a keyword query may be split across multiple tuples in different relations.

Masermann in [Masermann, 2000] introduced the keyword search over relational database, the main limitation of this work is that all keywords must be contained in the same tuple.

Discover model [Hristidis, 2002] solve this problem by returning qualified joining networks of tuples, that is, sets of tuples that are associated because they join on their primary and foreign keys, and collectively contain all the keywords of the query.

This model view a database as a data graph G that has tuples and keywords as nodes (see Figure (4.1)). Two tuples are connected by an edge if they can be joined using a primary to foreign key relationship, a tuple T and a keyword K are connected if T contains K. Thus, a result of a keyword search is a subtree of G that is reduced with respect to K. Ranking of results is based on the notion of keyword proximity; that is, a smaller reduced subtree has a higher rank.

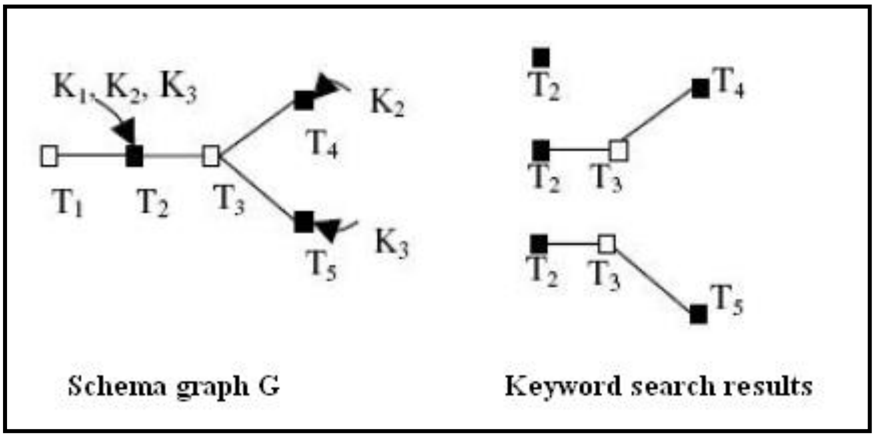


Figure 4.1: Keyword search results.

In this chapter we describe the ***Discover*** model in more details, which will be improved in the next chapter by allowing the user to associate the keyword term with a semantic term in the search string to help the computer to retrieve more relevant answers at first top- k.

To illustrate the different concepts of this chapter, we use a subset of TPC-H schema as shown in Figure (4.2) and an instance of it as shown in Figure (4.3).

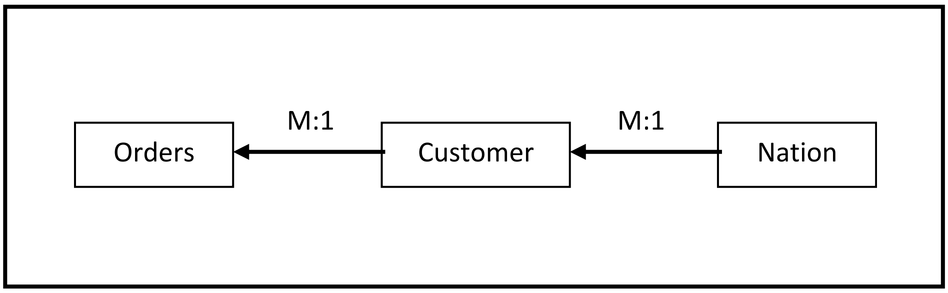
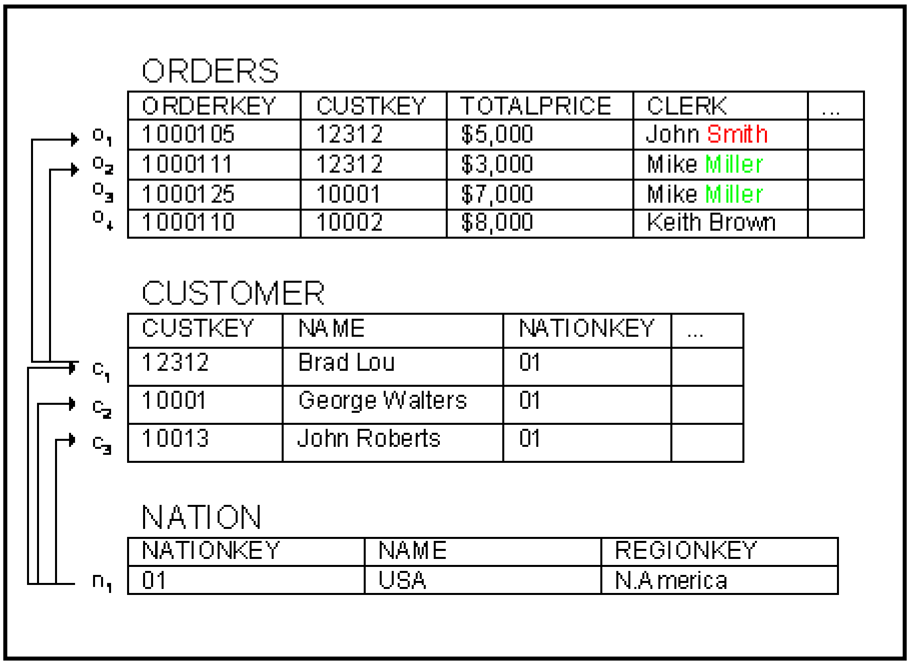


Figure 4.2: TPC-H schema [Hristidis, 2002].


ORDERS

	ORDERKEY	CUSTKEY	TOTALPRICE	CLERK	...
o_1	1000105	12312	$5,000	John Smith	
o_2	1000111	12312	$3,000	Mike Miller	
o_3	1000125	10001	$7,000	Mike Miller	
o_4	1000110	10002	$8,000	Keith Brown	

CUSTOMER

	CUSTKEY	NAME	NATIONKEY	...
c_1	12312	Brad Lou	01	
c_2	10001	George Walters	01	
c_3	10013	John Roberts	01	

NATION

	NATIONKEY	NAME	REGIONKEY
n_1	01	USA	N.America

Figure 4.3: Instance of TPC-H schema.

4.2 Discover Architecture

Figure (4.4) shows the ***Discover*** architecture, we applied the query string “Smith Miller” to the right side of the figure to illustrate the role of each module in generating the query answers. These modules will be described in more detail in sections 4.3 to 4.6.

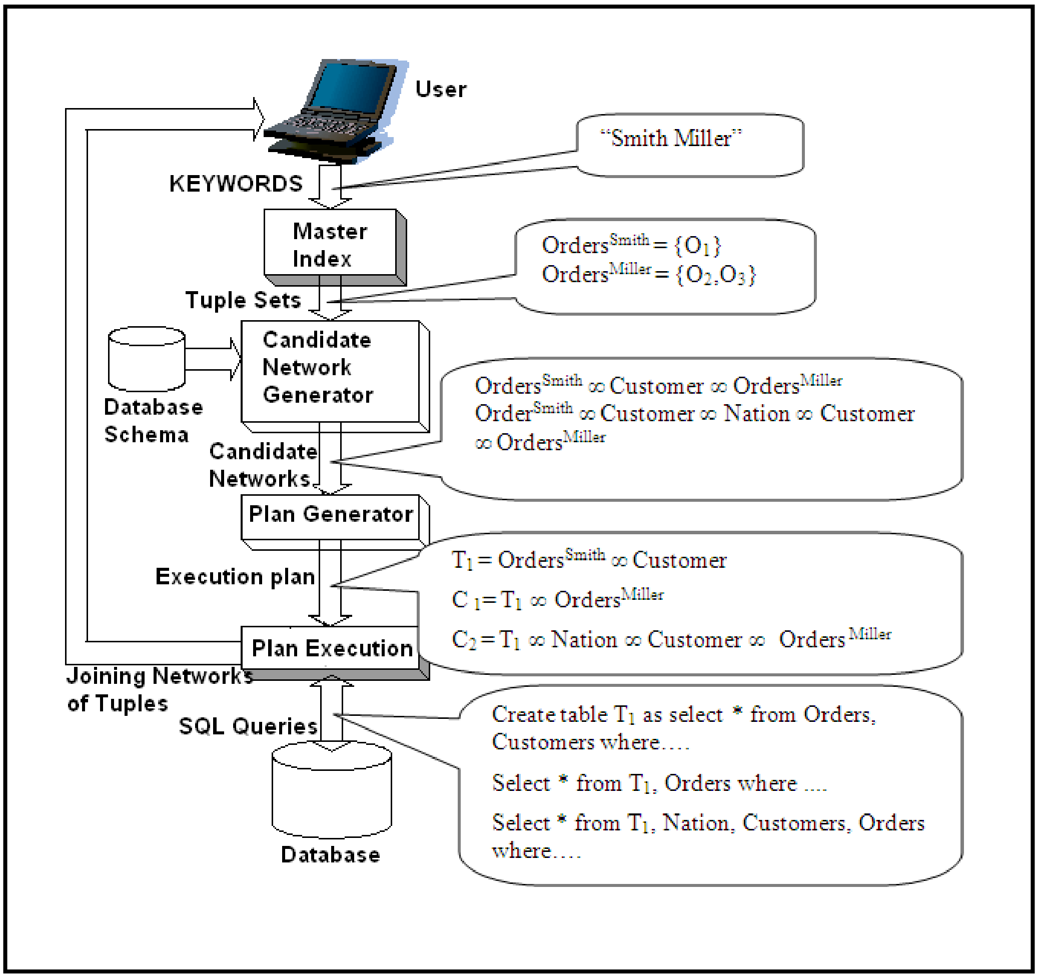

User
“Smith Miller”
KEYWORDS
Master Index
$\text{Orders}^{\text{Smith}} = \{O_1\}$
$\text{Orders}^{\text{Miller}} = \{O_2, O_3\}$
Tuple Sets
Candidate Network Generator
Database Schema
$\text{Orders}^{\text{Smith}} \infty \text{Customer} \infty \text{Orders}^{\text{Miller}}$
$\text{Order}^{\text{Smith}} \infty \text{Customer} \infty \text{Nation} \infty \text{Customer} \infty \text{Orders}^{\text{Miller}}$
Candidate Networks
Plan Generator
Execution plan
$T_1 = \text{Orders}^{\text{Smith}} \infty \text{Customer}$
$C_1 = T_1 \infty \text{Orders}^{\text{Miller}}$
$C_2 = T_1 \infty \text{Nation} \infty \text{Customer} \infty \text{Orders}^{\text{Miller}}$
Plan Execution
Joining Networks of Tuples
SQL Queries
Create table T_1 as select * from Orders, Customers where….
Select * from T_1, Orders where ….
Select * from T_1, Nation, Customers, Orders where….
Database

Figure 4.4: Discover Architecture.

The expression A ∞ B in Figure (4.4) denote that tuple A joins with tuple B on their primary to foreign key relationship.

The basic flow of information is as follows. The user enters a search query using a browser. The search Query Processor parses the query into a list of search terms. The Index Repository is used to fined nodes (tuples) that satisfy the search terms and to find out whether pairs of nodes are interconnected. Once the relevant information is returned the system creates the answers, which are ranked, sorted and then returned.

4.3 Search Query

Search queries in ***Discover*** contain one or more keywords separated by spaces. In response to these queries the results that actually contain all the search terms will return (AND semantics).

4.4 Master Index

This module builds full-text indices on single attributes using existing database system utilities. The Master Index inputs are a set of keywords entered by the user and the outputs are a set of tuples, where the tuple contains at least one of the user keywords.

For Example,

The user type the following query string “Smith Miller”

Input:

Two keywords Smith and Miller

Output:

O1, O2, O3

Where the tuple O1 is the only tuple in the searched database that contains the Keyword Smith, while the tuples O2 and O3 contain the keyword Miller.

4.5 Candidate Networks Generator

The candidate network is a set of tuples which have a primary to foreign key relationship and contains all the search terms, in the same time it is minimal which have at least one search term at each edge of the candidate network .Each candidate network represents a single answer.

The keyword may appear in multiple tuples, so the candidate networks can be too big, ***Discover*** determined the maximum size of the acceptable candidate network by the maximum number of tables stored in the original database.

For Example,

CN1: $O^{Smith} \leftarrow C \rightarrow O^{Miller}$

CN2: $O^{Smith} \leftarrow C \leftarrow N \rightarrow C \rightarrow O^{Miller}$

CN3: $O^{Smith} \leftarrow C \rightarrow O^{Miller} \leftarrow C$

CN1 and CN2 are acceptable candidate networks while CN3 is not acceptable, because it is not minimal (tuple C not contain any search term).

4.6 Execution Plan

Generated SQL queries are expensive due to joins, so this module finds the intermediate results that should be built, so that the overall cost of building these results and evaluating the candidate networks is minimum. After preparing SQL statements from the execution plan these statements will be executed over underlying database in descending order according to the ranking score.

For Example,

Each CN corresponds to a SQL statement

CN1: $O^{Smith} \leftarrow C \rightarrow O^{Miller}$

CN2: $O^{Smith} \leftarrow C \leftarrow N \rightarrow C \rightarrow O^{Miller}$

Execution Plan

$Temp \leftarrow O^{Smith} \infty C$

$CN1 \leftarrow Temp \infty O^{Miller}$

$CN2 \leftarrow Temp \infty N \infty C \infty O^{Miller}$

4.6.1 Ranking Algorithm

The ranking function takes into account the number of tuples involved in the joining network.

$$\boldsymbol{score(r) = \frac{1}{size(t_r)}}$$

Where:

r =the search query result.

t_r = the set of tuples that generate r .

size(t_r) = the number of tuples in $t_{r.}$

As the ranking score increase, the relevancy of the result increases accordingly.

4.7 Characteristics of Discover model

1. The search query is simple (list of keywords).

2. Provide irrelevant results at top- k.

3. Apply the results that satisfy AND semantics.

4.8 Summery

A ***search operation*** is initiated by typing the user the query syntax in the text box and then pressing the search button, ***Discover*** first parsing the query syntax by splitting the contents of the query string into tokens (words), then ***Discover*** retrieves all the matching records that have at least one of the user keywords, then ***Discover*** build all the different pairs combinations of tuples (matching records) to find the shortest path of each combination, the shortest path of each combination represent a candidate network, it ranks the candidate networks according to their ranking score in descending order which take into a count the length of the joining network, after applying the execution plan optimization algorithm over the candidate networks, it generates the equivalent SQL statement for each joining network and finally it executes the SQL statements over original database to retrieve the user's request results.

Chapter Five: Ssearch Model

5.1 Introduction

In the existing approaches of keyword search over relational databases, the user types a set of keywords to formulate the search query, then a set of records containing these keywords return as results without taking into account what user really means from typing these keywords (i.e. If the user is looking for the keyword Taha, the existing approaches can't distinguish what the user means by this keyword, does he mean the user Taha? or does he mean the author Taha? or books about Taha?), leading to a high number of irrelevant results at the first top-k, our proposed system solves this problem by giving a user an ability to be more accurate in expressing the search query by allowing the user to determine what he means from each searching keyword, by adding semantics (optional) for each typing keyword to improve the quality of the results (increase the number of relevant results at top-k).

In our proposed system, we will use some schema information (table name, field name) as a semantic term.

Ssearch stands for a semantic search engine over relational databases, in this system we modified the ***Discover*** model to apply our proposed idea, by generating a simple query syntax suitable for casual users and modifying the ranking function to take into account the user's semantics. In this chapter we will describe ***Ssearch*** model in details.

To illustrate the concepts and the outputs of the algorithms mentioned in this chapter, we created a simple example of a library database. Figure (5.1) shows the library database schema and Figure (5.2) shows an instance of the library database.

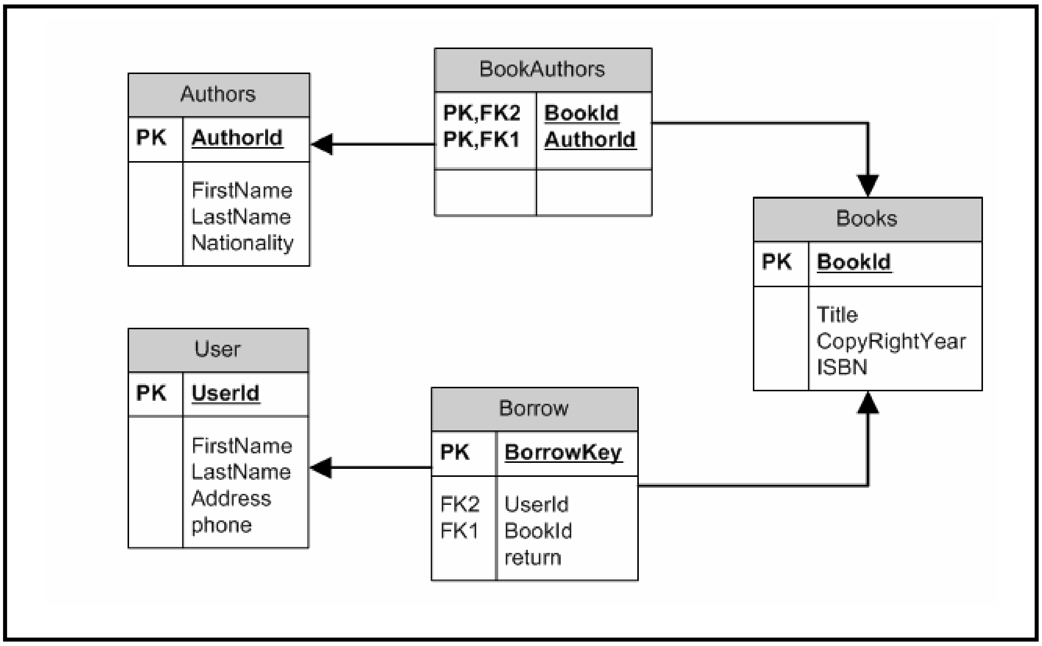
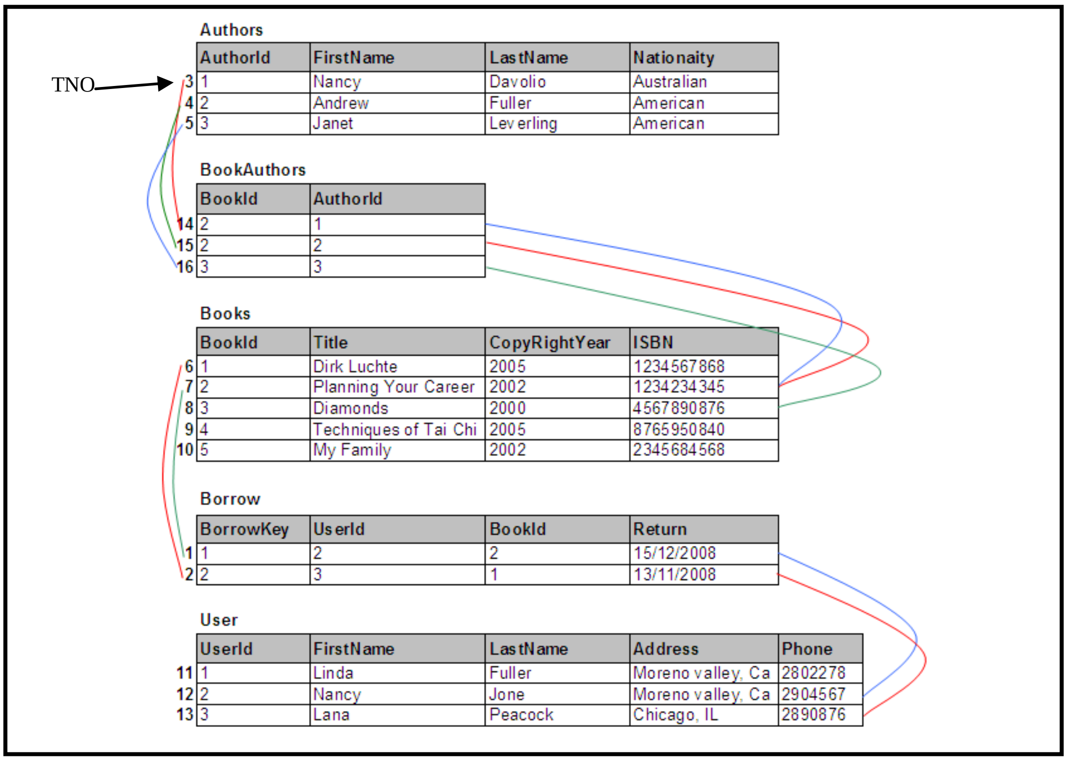


Figure 5.1: Library Schema



Authors

	AuthorId	FirstName	LastName	Nationaity
3	1	Nancy	Davolio	Australian
4	2	Andrew	Fuller	American
5	3	Janet	Lev erling	American

BookAuthors

	BookId	AuthorId
14	2	1
15	2	2
16	3	3

Books

	BookId	Title	CopyRightYear	ISBN
6	1	Dirk Luchte	2005	1234567868
7	2	Planning Your Career	2002	1234234345
8	3	Diamonds	2000	4567890876
9	4	Techniques of Tai Chi	2005	8765950840
10	5	My Family	2002	2345684568

Borrow

	BorrowKey	UserId	BookId	Return
1	1	2	2	15/12/2008
2	2	3	1	13/11/2008

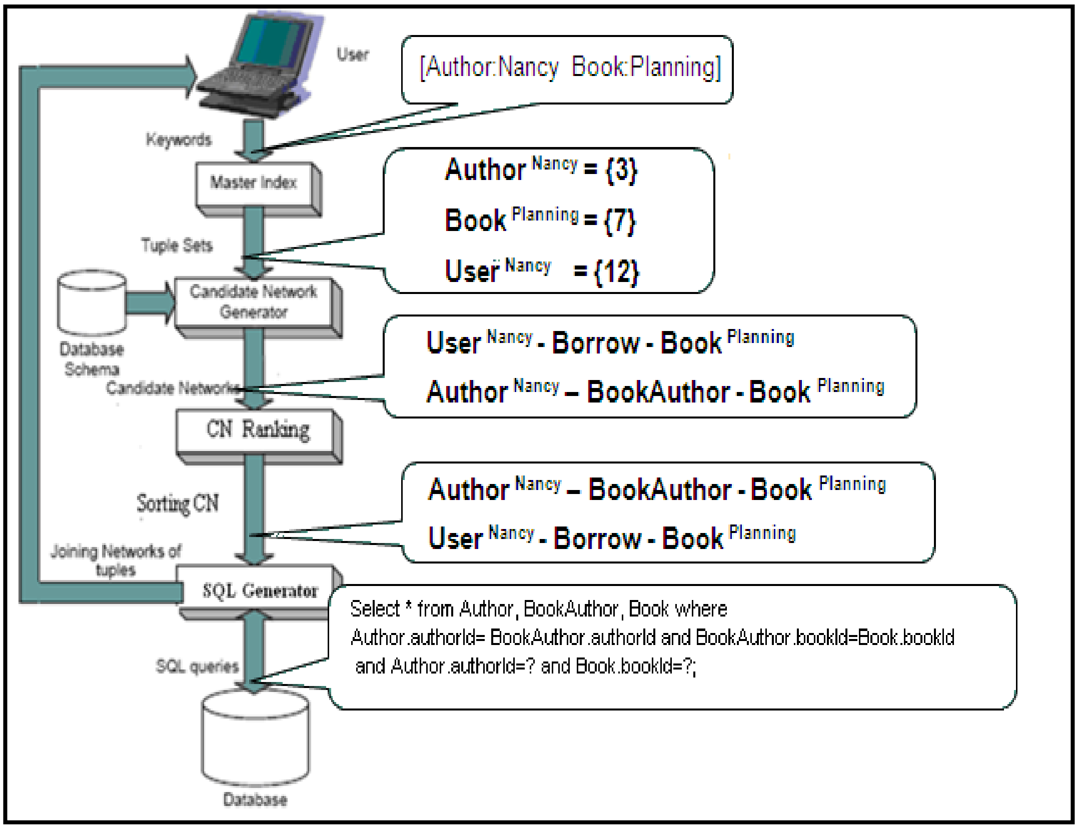
User

	UserId	FirstName	LastName	Address	Phone
11	1	Linda	Fuller	Moreno valley, Ca	2802278
12	2	Nancy	Jone	Moreno valley, Ca	2904567
13	3	Lana	Peacock	Chicago, IL	2890876

Figure 5.2: Sample of Library Instance.

5.2 Ssearch Architecture

As we mentioned above, the architecture of the proposed system depends on the architecture of ***Discover*** [Hristidis, 2002], with some modifications in the input search query, indexing and ranking algorithm to be suitable to our proposed idea. See Figure (5.3).


User
[Author:Nancy Book:Planning]
Keywords
Master Index
Tuple Sets
$\text{Author}^{\text{Nancy}} = \{3\}$
$\text{Book}^{\text{Planning}} = \{7\}$
$\text{User}^{\text{Nancy}} = \{12\}$
Database Schema
Candidate Network Generator
Candidate Networks
$\text{User}^{\text{Nancy}}$ - Borrow - $\text{Book}^{\text{Planning}}$
$\text{Author}^{\text{Nancy}}$ – BookAuthor - $\text{Book}^{\text{Planning}}$
CN Ranking
Sorting CN
$\text{Author}^{\text{Nancy}}$ – BookAuthor - $\text{Book}^{\text{Planning}}$
$\text{User}^{\text{Nancy}}$ - Borrow - $\text{Book}^{\text{Planning}}$
Joining Networks of tuples
SQL Generator
SQL queries
Database

```
Select * from Author, BookAuthor, Book where
Author.authorId= BookAuthor.authorId and BookAuthor.bookId=Book.bookId
 and Author.authorId=? and Book.bookId=?;
```

Figure 5.3: Ssearch Architecture.

The basic flow of information is as follows. The user enters a search query using a browser. The search Query Processor parses the query into a list of pairs, semantic term and search term. The Index Repository (Master index) is used to find nodes (tuples) that satisfy the search terms, verify if the search terms within the node satisfy the user semantic or not and find out whether pairs of nodes are interconnected. Once the relevant information is returned the system creates the answers, which are ranked, sorted and then returned.

Ssearch system is divided into two main subsystems:

1- Offline system, which is related to all the operations needed to build and update the master index database.

2- Online system, which is related to all the operations needed to process the user's request.

5.3 Offline System

As mentioned before, Offline System operations are all the processes needed to build and update the master index database, which collects, parses, and stores data to facilitate fast and accurate information retrieval.

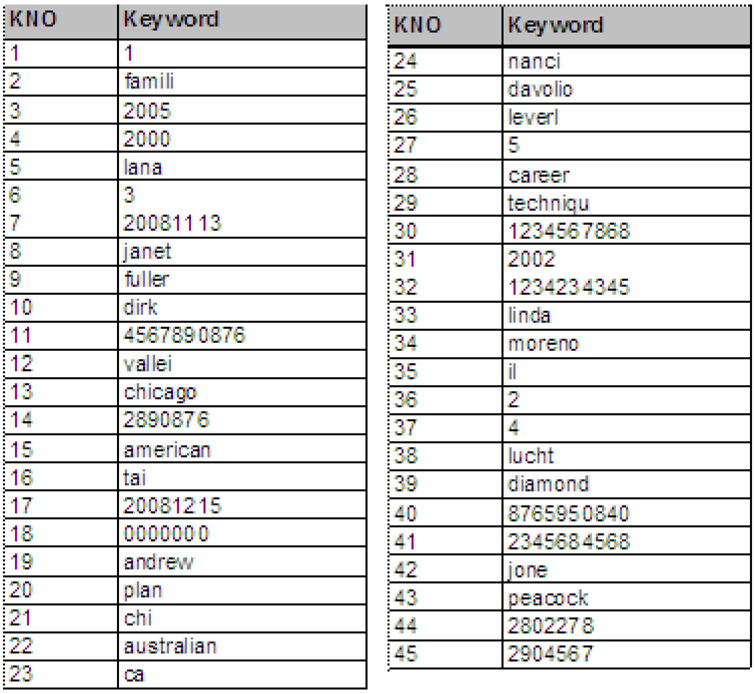
The Master Index database (or a desired part of it) is enabled for keyword search through the following steps. Initially a copy of all the original tables and its contents will be saved in the master index database, then required tables are created for supporting keyword searches, which is used at search time to efficiently determine the locations of query keywords in the database (i.e., the tables, columns, rows they occur in). In the Master Index database we create four tables to support the search operations in an efficient way.

1- Keywords Table

This table contains all the words (keywords) that appear in the original database after removing stop words and stemming, we identify a unique identifier for each keyword to distinguish between them. For example, see Table (5.1) which shows the contents of the keywords table for the Library database.

Ssearch uses this table to determine if the search term (keyword term) after stemming appears in the original database or not. We improve the search time by creating an index on the keyword column.

Table 5.1: Keywords table of the Library DB.



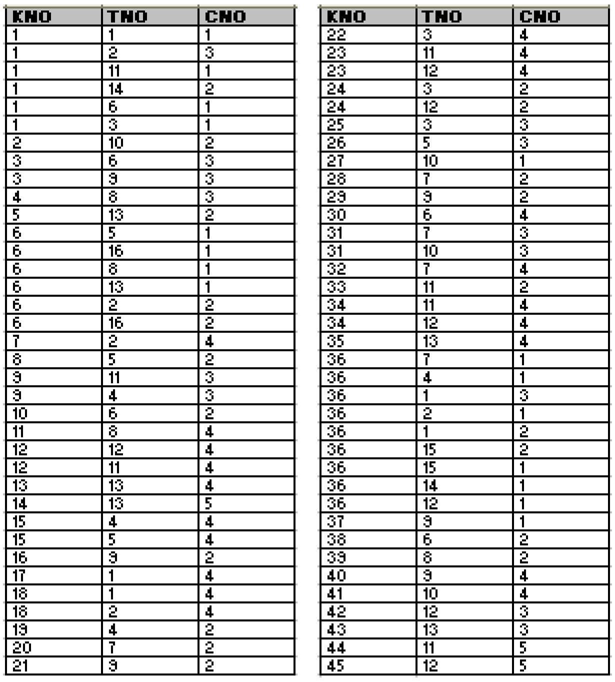
KNO	Keyword
1	1
2	famili
3	2005
4	2000
5	lana
6	3
7	20081113
8	janet
9	fuller
10	dirk
11	4567890876
12	vallei
13	chicago
14	2890876
15	american
16	tai
17	20081215
18	0000000
19	andrew
20	plan
21	chi
22	australian
23	ca

KNO	Keyword
24	nanci
25	davolio
26	leverl
27	5
28	career
29	techniqu
30	1234567868
31	2002
32	1234234345
33	linda
34	moreno
35	il
36	2
37	4
38	lucht
39	diamond
40	8765950840
41	2345684568
42	jone
43	peacock
44	2802278
45	2904567

2- Keywords Information Table

This table stores information about the locations of each keyword in the database (the row number (TNo) and the column number (CNo) where the keyword appears), the same keyword may appear in more than one location in the database. (i.e., the keyword Nancy appears in two locations (See Table (5.2)), the first one appears in the tuple number 3 and in the second column of the row, the second one appears in the tuple number 12 and in the second column of the row). Table (5.2) shows the contents of the Keywords Information table for the Library DB.

Table 5.2: Keywords Information table of the Library DB.



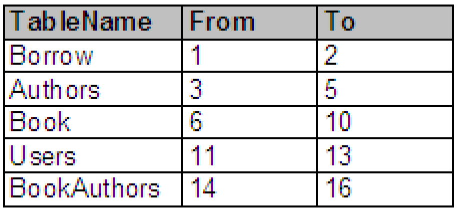
KNO	TNO	CNO
1	1	1
1	2	3
1	11	1
1	14	2
1	6	1
1	3	1
2	10	2
3	6	3
3	9	3
4	8	3
5	13	2
6	5	1
6	16	1
6	8	1
6	13	1
6	2	2
6	16	2
7	2	4
8	5	2
9	11	3
9	4	3
10	6	2
11	8	4
12	12	4
12	11	4
13	13	4
14	13	5
15	4	4
15	5	4
16	9	2
17	1	4
18	1	4
18	2	4
19	4	2
20	7	2
21	9	2
22	3	4
23	11	4
23	12	4
24	3	2
24	12	2
25	3	3
26	5	3
27	10	1
28	7	2
29	9	2
30	6	4
31	7	3
31	10	3
32	7	4
33	11	2
34	11	4
34	12	4
35	13	4
36	7	1
36	4	1
36	1	3
36	2	1
36	1	2
36	15	2
36	15	1
36	14	1
36	12	1
37	9	1
38	6	2
39	8	2
40	9	4
41	10	4
42	12	3
43	13	3
44	11	5
45	12	5

3- Tuples Information Table (tuplesInfo)

Ssearch use this table to map between the tuple number and its corresponding table name. Table (5.3) shows the range of the tuples numbers of each table in the Library DB.

After stemming the search terms, ***Ssearch*** searches the keyword term in the Keywords table, if the keyword is found, it uses the corresponding term identifier to retrieve the location of the keyword in the DB (Tno, Cno) from the Keyword Information table, then by using the Tno it retrieves the table name from the Tuples Information table, and then by using the table name and the column number it can retrieve the column name using the database catalog, ***Ssearch*** uses both table name and column name to check if the keyword term in specific record satisfies the user’s semantics or not (if the semantic term matches the table name or the column name of the keyword term then we can say that the keyword term satisfies the user’s semantic at this record).

Table 5.3: Tuples Information table of the Library DB.



TableName	From	To
Borrow	1	2
Authors	3	5
Book	6	10
Users	11	13
BookAuthors	14	16

4- Primary to Foreign key table (PkFk_Table)

This table contains information about the foreign to primary key relationships between records (tuples).

Each record in this table stores the following information:

- A pair of tuple numbers that have a primary to foreing key relationship (PTNo, FTNo).

- The share value between the pair of related tuples.

- The primary column name and the foreign column name in the relationship.

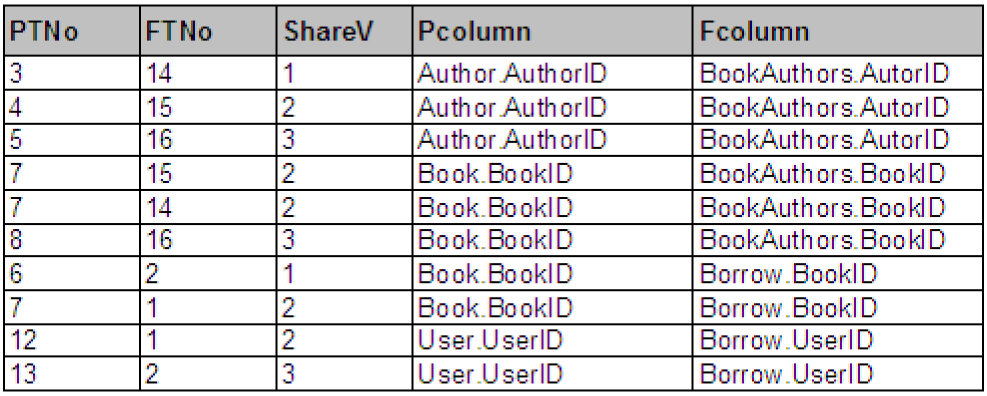
For example, See Table (5.4), tuple 3 and tuple 14 have a primary to foreign key relationship, because tuple 3 refers to tuple 14 by the share value 1 in the authorId column which is a primary key column in Author table and a foreign key column in BooksAuthor table.

We will use the contents of this table later in two cases:

- To build the data graph of the original database.

- To generate the Where part of the SQL statement.

Table 5.4: Primary to Foreign key table of the Library DB.



PTNo	FTNo	ShareV	Pcolumn	Fcolumn
3	14	1	Author.AuthorID	BookAuthors.AutorID
4	15	2	Author.AuthorID	BookAuthors.AutorID
5	16	3	Author.AuthorID	BookAuthors.AutorID
7	15	2	Book.BookID	BookAuthors.BookID
7	14	2	Book.BookID	BookAuthors.BookID
8	16	3	Book.BookID	BookAuthors.BookID
6	2	1	Book.BookID	Borrow.BookID
7	1	2	Book.BookID	Borrow.BookID
12	1	2	User.UserID	Borrow.UserID
13	2	3	User.UserID	Borrow.UserID

5.3.1 Offline System Algorithms

This section describes the algorithms needed to build the master index database, we will describe these algorithms in order of their execution.

5.3.1.1 Data Gathering Algorithm

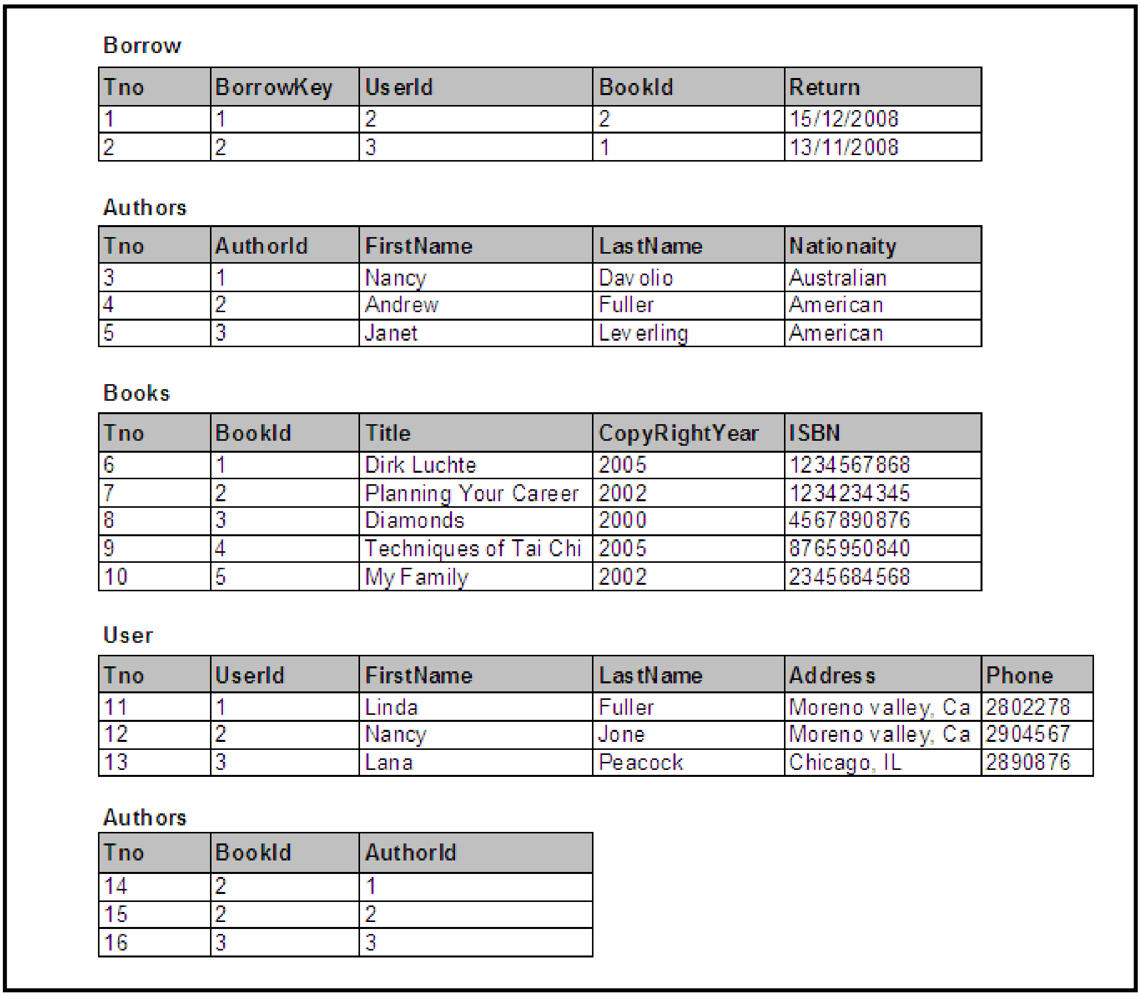
This algorithm copies the contents of the user tables from the original DB into the local DB (Master Index DB)

Algorithm description:

- for each user table in the original DB
  - creates a table with the same table and columns name in the local database
  - copies the contents of the original table into the local table
  - alters the local table by adding new column (Tno) for inserting the tuple No.
  - in the new column , insert a unique value for each row in the created table (which is a unique value for each record copied to the local database)

Output:

Figure (5.4) shows the output of the algorithm.



Borrow

Tno	BorrowKey	UserId	BookId	Return
1	1	2	2	15/12/2008
2	2	3	1	13/11/2008

Authors

Tno	AuthorId	FirstName	LastName	Nationaity
3	1	Nancy	Dav olio	Australian
4	2	Andrew	Fuller	American
5	3	Janet	Lev erling	American

Books

Tno	BookId	Title	CopyRightYear	ISBN
6	1	Dirk Luchte	2005	1234567868
7	2	Planning Your Career	2002	1234234345
8	3	Diamonds	2000	4567890876
9	4	Techniques of Tai Chi	2005	8765950840
10	5	My Family	2002	2345684568

User

Tno	UserId	FirstName	LastName	Address	Phone
11	1	Linda	Fuller	Moreno valley, Ca	2802278
12	2	Nancy	Jone	Moreno valley, Ca	2904567
13	3	Lana	Peacock	Chicago, IL	2890876

Authors

Tno	BookId	AuthorId
14	2	1
15	2	2
16	3	3

Figure 5.4: The local copy of the Original DB.

We generate a copy of the original database tables into the local database for two reasons:

1- To improve the execution time needed for updating the Master Index Database by:

- Parsing the data stores in the original database locally (the process of analyzing a text, made of a sequence of tokens (words), for determining the different keywords stored in the database), to decrease the transfer time through a network.

  - Generating the primary to foreign key table locally, leads to a critical improvement in the execution time by comparing with the same operation over remote (original) database.

2- To add the cash property to the ***Ssearch*** model (like Google). This gives the user an opportunity to see the contents of the most recent copy of the original database in the master index database, if the data in the original database is removed or updated after indexing.

5.3.1.2 Build the Keywords Information table Algorithm

Algorithm description:

- Create a table (KeywordInfo) having the following fields : Keyword term, TupleNo, ColumnNo

- For each table in the local database (Master Index database)

  - Parse the contents of each record in the table

  - For each token (keyword) in the record after stemming and removing the stop words

    - Insert new record in the keyword information table contains the following information : stemming keyword, the record identifier (tuple no.) and the column number where the keyword appear

Output:

Table (5.2) shows output of the algorithm after performing on the Library DB.

5.3.1.3 Build the Keywords table algorithm

Algorithm description:

- Create a table (Keywords) having the following fields : Keyword Identifier(KNo), keyword term
- Use SQL query to retrieve all distinct keywords from the keywords information table
- Insert these keywords in the Keywords table
- Associate each keyword with a unique identifier
- Replaces the keyword term column in the KeywordInfo table with its corresponding identifier

Output:

Table (5.1) shows the Keywords table and Table (5.2) shows Keywords Information table after replacing the keyword term column with the corresponding keyword number (Kno).

5.3.1.4 Build the Primary to Foreign key table Algorithm

Algorithm description:

- Creates a table (PkFk_Table) having the following fields: primary tuple identifier (PTNo), foreign tuple identifier (FTNo), share value (ShareV), Primary column name(PColumn) and foreign colomn name (FColumn)
- For each table in the original database which has at least one foreign key column
    - For each foreign key column in the foreign table

- Retrieves the primary table name and the primary column name that the foreign key column refers to.

- Use the primary table name, primary column name, foreign table name and foreign column name to retrieve the matching records between the primary table and foreign table (retrieve the tuple number of the primary table,the tuple number of the foreign table and the share value) from the local database.

- append the matching records information to the PkFk _Table

Output:

Table (5.4) shows output of the algorithm for the Library DB.

5.3.2 Updating the Master Index Database

Updating the contents of the Master Index database can be done automatically without any custom configurations. We need only to establish a connection with the original database by specifying the following parameters: the user's name and password for a user which has a privilege to access the original database, the database name and the IP address of the server which host the original database.

The master index database must be periodically updated with the latest information, it could be done weekly, monthly or yearly depending on the amount of transactions that happen over the original database. The aim of this operation is to provide the user with up-to-date results.

The ***Ssearch*** retrieves the query answers from the database by generating SQL statement for each joining network and executing it over the original database, or by using the ***Ssearch*** cash which retrieves the query answers from the local database which have the most recent copy of the original database, such as Google cash.

Ssearch cash can provide the user with information not longer available in the original database or with information not yet updated in Master Index DB.

The results of the user query depend on the contents of the master index database.

5.4 Online System

Online System operations are all the operations needed to process the user request, which analyze the query syntax, matching between the keyword term and the record number that contains it, verify if the matching record satisfies the user semantic term or not, generate the joining networks, ranking the results according to relevancy, generate SQL statement equivalent for each joining network and finally execute the SQL statements in order to retrieve the query answers.

The following sections describe these operations in detail.

5.4.1 Suggested query syntax

A search term has the form l:k, l: or k where l: is a label and k is a keyword, the labels give semantics to the keyword terms, the label may be table name or field name.

The semantic term in our system (***Ssearch)*** can be applied but not required, if the user enters more than one search term, the ***Ssearch*** system will retrieve only those results that contain all the terms coming from the search request (AND semantic). ***Ssearch*** system allows the user to enter up to 20 search terms.

For example, This example shows how the semantic term can improve the relevancy of the results at top-k, assume the user type the search query “Nancy Planning” over the Library database, see Figure (5.1) and Figure (5.2) where the links in Figure (5.1) point to the direction of the foreign to the primary key (many to one) relationships between tables. According to the current search engine ***Discover***, the results of the query will be two minimal joining networks that contain the both keywords Nancy and planning, 12 ∞ 1 ∞ 7 and 3 ∞ 14 ∞ 7, we use the expression a ∞ b to denote that tuple a joins with tuple b on their primary to foreign key relationship, each one of the symbols a and b refers to a different tuple (record) in a table, we identified each tuple in the Library instance (see Figure (5.2)) by a unique number for simplicity. The two results in the current search engines have the same score (rank), because the two keywords appear in the both results and the two minimal joining networks have the same size (size= 3), If we take into account the meaning of each result, we will discover that each result has completely different meaning than other one, the first result 12 ∞ 1 ∞ 7 shows that the user Nancy Jone borrowed "Planning your career" book from the library, whereas the second result 3 ∞ 14 ∞ 7 shows information about the "Planning your Career" book where Nancy Davolio is one of its authors. Imagine that we apply this keyword search query to a huge database, as a result of that, most of the top-k results will be irrelevant to what the user really means, so by using our suggested query syntax, the user can reformulate the keyword search query in more accurate way by adding semantic term to each keyword term to get more relevant results at first top-k. Using our suggested query syntax , the user can refine the query string to be "User:nancy book:planning", if he searches for information about the user Nancy, who borrowed books about planning, which will return two minimal joining networks that contain the both keywords Nancy and planning, 12 ∞ 1 ∞ 7 and 3 ∞ 14 ∞ 7 , but with different scores depending on the semantics of the search query, which will display the minimal joining network 12 ∞ 1 ∞ 7 at the top of the search results , also the user can refine the query string to be "Author:nancy book:planning", if he searches for information about the author Nancy who writes books about planning, which will return two minimal joining networks 3 ∞ 14 ∞ 7 and 12 ∞ 1 ∞ 7, which will display the first one at the top of the search results, in both cases the most relevant result will be display at the top .

Ssearch system retrieves the same number of results as the previous system (***Discover***) but with different ranking score, which take into account the user semantics.

5.4.1.1 Query Examples

The sample queries provided in this section are based on the sample data model (see Figure (5.1), Figure (5.2)).

Table (5.5) shows different examples of query string that can be applied in ***Ssearch*** model to understand the power of the suggested query syntax.

Table 5.5: Examples of the suggested query syntax.

Query Syntax	
Query	**Description**
Nancy planning	Returns all related records that contain the two keywords Nancy and planning
Author:	Returns all the author's table records
Author: Andrew	Returns all the authors whose name contain "Andrew"
Author: book: planning	Returns all related records that have information about the authors and the books of planning
Books:	Returns all the books table records
Books: Author:	Returns all related records that contain information about the books in the library and their authors
Books: user: Nancy	Returns all the books borrowed by the user Nancy
Author:Nancy book:planning	Returns all the books of the author Nancy in the domain of planning
ISBN : 4567894039	Returns the book information where the ISBN equals 4567894039

We notice from the above examples that the suggested query syntax is not only the syntax that allow the user to apply semantic term for the search keyword (To make the computer distinguishes the true meaning of the Keyword), but it also allows the user to write queries which provides the user with ***for-all*** semantics results (i.e. The query string "books:" retrieves all the contents of the book's table).

The traditional keyword search engines over relational DB lack of the previous characteristics.

5.4.2 Parsing the Query String Algorithm

Suppose a user enters the following search criteria:

Author:Nancy Book:planning

This query must return all the books of the author Nancy in the domain of planning.

We will use this query to explain all the operations will be done over it, to retrieve the final results.

Algorithm description:

- For each token in the query string after stemming and removing stop words
  - If the suffix of the token is ':' then
    - Consider the token as a semantic term
  - Else
    - Consider the token as a keyword term
  - If the system reads a keyword term after a semantic term then
    - Link between the keyword term and the semantic term
  - Else if the system reads a keyword term after a keyword term then
    - The keyword term has no semantics
  - Else if the system reads a semantic term after a semantic term then
    - The first semantic term has no corresponding keyword term

Output:

The output of the parsing algorithm is stored in two dimensional matrix (Parsing Matrix):

Index	Semantic	Keyword
0	Author	Nanci
1	Book	Plan

Figure 5.5: Output of the Parsing Algorithm (Parsing Matrix).

Where each row of the matrix contains both the semantic term and its corresponding keyword term, or one of them if the other is absent.

5.4.3 Retrieving Matching Records Algorithm

After parsing the query string, the system retrieves the tuples numbers that match the keyword term in each row of the parsing matrix if found, else the row of the parsing matrix contains only the semantic term, so the system assumes that all the tuples that satisfy the semantic term matches this row (i.e. the output of parsing the query string “Author:” will be one row in the parsing matrix that have only the semantic term ‘author’, this semantic term matches the following tuples : 3, 4 and 5).

The tuple satisfy the semantic term if the tuple table name or one of its columns name have the same name as the semantic term after stemming.

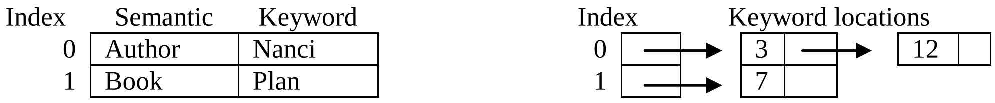
Algorithm description:

- For each row in the parsing matrix
    - If the row has keyword term then
        - Retrieve all the tuples numbers that contains it
    - Else
        - Retrieve all the tuples numbers that satisfy the semantic term

Output:

The output of this algorithm store in an array of a linked list to save the memory space because the system can't predict how many times the particular keyword appears in the DB (the probability of the total number of matching records increase, if the total number of a particular keyword that appear in the database increase).

The following figure, shows the output of the algorithm.



Input (Parsing Matrix)

Output(Matching Records)

Figure 5.6: The Output of Retrieving Matching Records Algorithm.

The array index number of the matching records and its particular keyword are the same.

5.4.4 Semantic Satisfaction Algorithm

This algorithm check if the matching record of a particular keyword satisfies the user's semantic or not.

The semantic satisfaction occurs if the matching record satisfies one of the following conditions:

1. If the table name of the matching record equals the user's semantic term.

2. If the column name of the matching record where the particular keyword appear equal the user's semantic term.

3. If the user didn't assign a semantic name to the keyword term, in this case, the user does not have a problem whenever the keyword appears in the database,

so the system assumes that the matching record of a particular keyword satisfies user ‘s semantic whenever it appears in the database.

4. If the user didn’t assign a keyword term to the semantic term, in this case, all the matching records (which are all the table records that have the same name as the semantic term or all the records where the semantic term has the same name as one of its field names) satisfies user’s semantic.

Algorithm description:

- For each row in the parsing matrix
    - If the current parsing matrix row has the both terms (semantic and keyword term)
        - For every matching record
            - If the matching record table’s name or column’s name equal the semantic term then
                - The matching record satisfies the user’s semantic
            - Else if the current parsing matrix row has only one term (semantic term or keyword term)
                - The matching record satisfy user’s semantic
            - Else
                - The matching record not satisfy the user’s semantic

Output:

The output of the above algorithm store in array of linked list, this structure store the semantic satisfaction status for each matching record in its corresponding location, where the symbol ‘T’ means that the matching record satisfies the user’s semantic while the symbol ‘F’ means that the matching record does not satisfy the user semantic.

The following figure, shows the output of the algorithm.

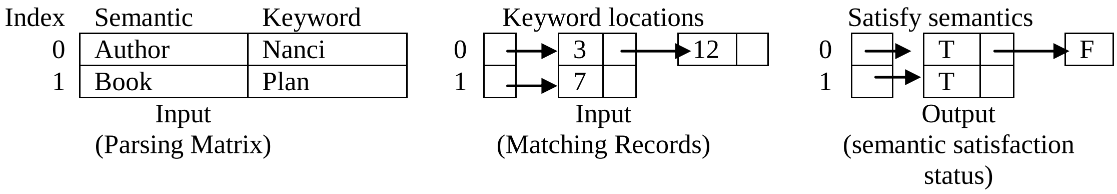


Figure 5.7: The Output of the Semantic Satisfaction Algorithm.

5.4.5 Tuples Combinations

In this section the system need to find all possible combinations of the set of tuples that matched the search terms, each combination will be considered as a pair (S,D) where S is the source node, D is the destination node in graph, to retrieve later in this chapter all the possible solutions of the user search query, by finding the shortest path of each pair. *Ssearch* will represent the data graph by using undirected graph, so the path between (x,y) is equivalent to the path between (y,x), consequently the order of the two elements in a pair is not important.

For example, depending on the matching records of Figure (5.6)

Let S refers to the set of tuples that matched the search terms and let T refers to the set of tuples combinations. Then:

S= {3, 7, 12}

and

T= {{ 3 , 3}, {3 , 12}, {3 , 7} , {12 , 12}, {12 , 7}, {7 , 7}}

Algorithm Description: (Tuple-Combinations Generator)

```
• Generate one-dimensional array (matchTuples) which contains all the matching
  records after removing duplicates
• n= number of elements in matchTuples array
• rowNumber=0
• for i=0 to n-1
    • for j=i to n-1
        • tuplePairs[rowNumber][0]=matchTuple[i]
        • tuplePairs[rowNumber][1]= matchTuple[j]
        • rowNumber = rowNumber+1
```

Output:

All the possible pairs of tuples store in two dimensional matrix (tuplePairs). See Figure (5.8).

Index	**Source**	**Destination**
0	3	3
1	3	12
2	3	7
3	12	12
4	12	7
5	7	7

Figure 5.8: The output of the Tuple-Combinations Generator Algorithm (TuplePairs matrix).

5.4.6 Data Graph

Ssearch represents the database content as a graph where the nodes are the database tuples and the edges are relationships among tuples.

Figure (5.9) shows the data graph of the database instance in Figure (5.2).

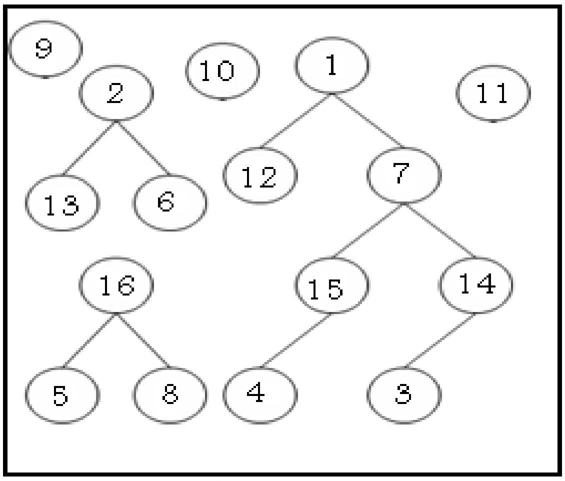


Figure 5.9: Graph of the database instance in Figure (5.2).

The Data Graph of ***Ssearch*** is undirected graph where each pair of tuples T_1 and T_2 such that there is a foreign key from T_1 to T_2, the graph contains an edge from T_1 to T_2 and a back edge from T_2 to T_1.

5.4.6.1 Data Graph Representation Algorithm

The data graph is represented using adjacency list, which is an array of linked list, to save the memory space because the data graph of any database in general is sparse graph.

We use the contents of the Primary to Foreign key table in the master index database (see Table (5.4)) to retrieve the pairs of tuples that have primary to foreign key relationship to build the adjacency list, the first element of the pair is stored in the PTNo field while the second one is stored in FTNo field.

Algorithm Description:

Input: the Primary to Foreign key table (PkFk_Table)

- For each record in the PkFk_Table
    - Retrieve the current PTNo and FTNo value
    - Append the value of PTNo field to the linked list tail which is found at the location LinkeList[FTNo]

Output:

The output of the algorithm will be adjacency list as shown in Figure (5.10) which represents the data graph in Figure (5.9).

For more illustration, Table (5.6) shows the pairs of tuples in the master index DB, that have primary to foreign key relationship which is the only information needed to represent the data graph.

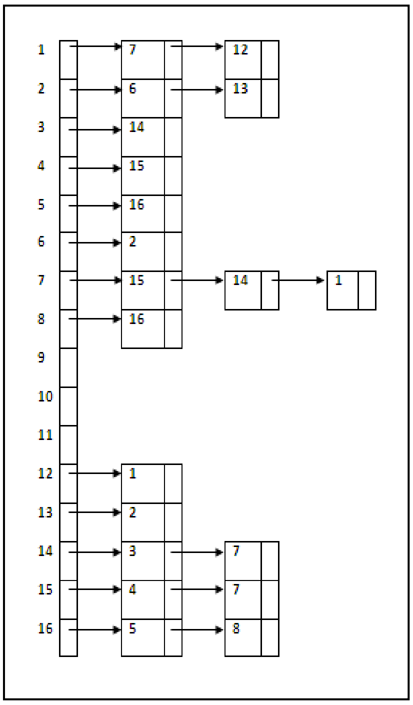
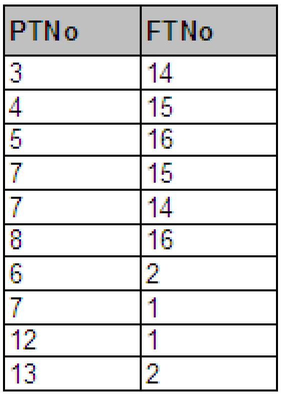


Figure 5.10: The output of the Data Graph representation using Adjacency List.

Table 5.6: Pairs of tuples that have primary to foreign key relationship.



PTNo	FTNo
3	14
4	15
5	16
7	15
7	14
8	16
6	2
7	1
12	1
13	2

5.4.7 Candidate Network Generator Algorithm

In this section ***Ssearch*** will find all possible solutions for the user request (query), by finding the shortest path between the pair of tuples in the TuplePairs matrix (Figure (5.8)), where each tuple contains at least one keyword term or satisfies at least one of the user's semantics. The resulted candidate networks are minimal (the start point and the end point of the candidate network have at least one of the user's keyword terms or satisfies at least one of the user's semantics, while the intermediate points not necessary to satisfy any of them, the intermediate points only needed to find the set of tuples that joins between the two end points).

Algorithm Description:

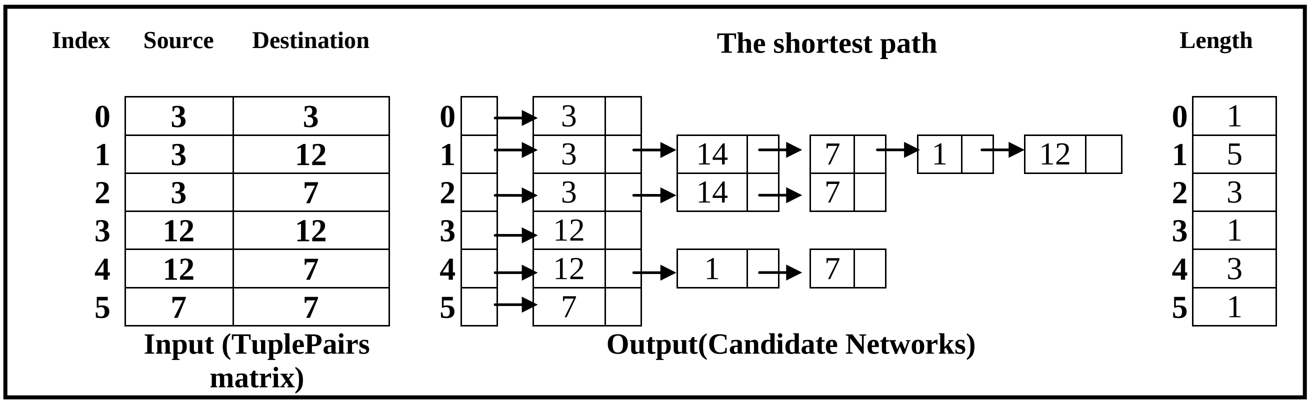
Input: TuplePairs Matrix (see Figure (5.8))

Assume the first element in the pair is the source node and the second element is the destination node

- For i=0 to last row index number of the tuplePairs matrix
    - Find the shortest path between TuplePairs[i,0] and TuplePairs[i,1] using Dikjestra algorithm
    - For each node in the shortest path
        - Append the node to the tail of the liked list, which is found at the location i of the array of linked list

Output:

The output of the algorithm is array of linked list which contains the shortest path of all the possible solutions. The following figure represents the output of the algorithm



Index	Source	Destination
0	3	3
1	3	12
2	3	7
3	12	12
4	12	7
5	7	7

Input (TuplePairs matrix)

	The shortest path	Length
0	3	1
1	3 → 14 → 7 → 1 → 12	5
2	3 → 14 → 7	3
3	12	1
4	12 → 1 → 7	3
5	7	1

Output(Candidate Networks)

Figure 5.11: Output of the candidate network generator algorithm.

5.4.8 Pruning Candidate Networks Algorithm

Not all the possible solutions are relevance to the user request.

Ssearch pruning the following candidate networks:

- Disconnected path which is a pair of nodes that is not connected in any way in the data graph (the length of the path is zero).

- The path with length more than the total number of tables in the original database (in this case the candidate network generate an answer with a weak meaning). In our experience of normalization, we have noticed that the path length between any two tables in the ER- Diagram can be at maximum the total number of tables in the database -1.

- The path that does not contains all the user keyword terms, ***Ssearch*** retrieve the answers that satisfy AND semantics only.

Output:

Figure (5.12) shows the remaining candidate networks (joining networks) after pruning the candidate networks in Figure (5.11).

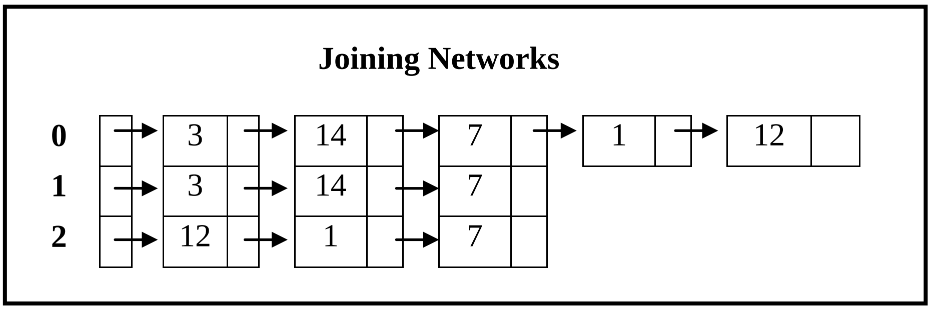


Figure 5.12: The candidate networks after pruning.

5.4.9 Ranking Algorithm

After pruning the candidate networks, ***Ssearch*** ranks the joining networks in descending order according to the ranking score using Quick sort algorithm.

As the ranking score increase, the relevancy of the result increases accordingly.

The suggested ranking function takes into account two factors: the user's semantics and the number of tuples size($t_{r)}$ involved in the joining network r.

$$score(r) = average\left(\frac{1}{size\,(t_r\ )} \quad , \quad \frac{\sum_{i=1}^{n_r} s(k_i\,)}{n_r} \right)$$

Where:

r =the search query result (the joining network).

t_r = the set of tuples that generate r .

size(t_r) = the number of tuples in $t_{r.}$

n_r = the total number of keywords in r .

k_i= the i-th keyword in the search query.

$$s(k_i) = \begin{cases} 1, & k_i\ satifies\ the\ \ semantic\ term \\ 0, & k_i\ not\ satisfies\ the\ semantic\ term \end{cases}.$$

Output:

See Figure (5.13).

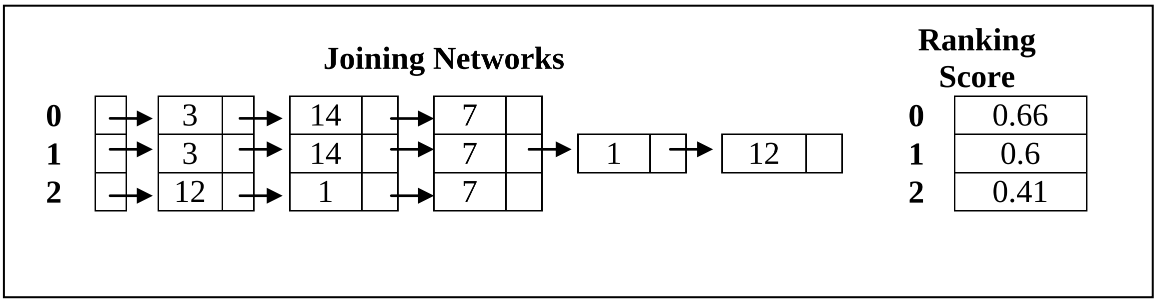


Figure 5.13: The order of the joining networks after ranking and their corresponding ranking scores.

5.4.10 SQL Answers

After retrieving the joining networks the model generate an equivalent SQL statement for each joining network to execute the SQL statements over the original database to retrieve up-to-date results, and over the master index database to retrieve the last version of answers before updating.

The following points should be considered when generating the SQL answers:

- Some columns may have the same names in different tables.
- Some nodes in the joining network may come from the same table.
- The SQL statement must retrieve only one answer.

To solve the problem of repeating the column’s name or the table’s name, the model assigns different alias to each table in the FROM part of the SQL statement where each node in the joining network will be assigned with its own table name.

To retrieve only one result from each joining network, the model adds conditions in the WHERE part of the SQL statement to determine the intended record from each table, where the collection of those records will formulate the answer. If the node in the joining network locate in the primary part of the relation between two nodes, the condition will retrieve only one result using the primary key value of this node, but if it locate in the foreign part of the relation, the condition may retrieve more than one result using the foreign key value of this node, therefore to be sure that the model will retrieve only the intended record, the model retrieve the primary key value of the foreign node to be used in the WHERE condition.

Algorithm Description (SQL answer generator):

There are two cases for generating the SQL statement:

- Case 1: If the answer stored in one table.

- Case 2: If the answer stored in more than one table.

Assume the tuple numbers in the joining network have the following variable names t_0, t_1, t_2, t_3…etc., where t_0 is the first node in the joining network, t_1 the second node and so on…

Case 1: (The answer stored in one table)

1. Generate the SELECT and the FORM part of the SQL statement
    - Retrieve the table name of t_0 from tuplesInfo table (see Table (5.3)) using the following SQL statement:

      ```
      select tableName
      from  tuplesInfo
      where t0 between from and to
      ```

    - s = “select * from ” + tableName

2. Generate the Where part of the SQL statement

    - Retrieve the primary key value of t_0

      To retrieve the intended record (t_0) from the database, use the database catalog to retrieve the primary key column name of the table, then by using the table name and the primary key column of the table, execute the following SQL statement over the local DB, to retrieve the primary key value of t_0 :

```
select primaryKeyColumn
from tableName
where tNo =t0
```

- w= “ where “ + primarykeyColumn + “=” + primaryKeyValue

3. Generate the complete SQL statement

    sql = s + w

Case 2: (The answer stored in more than one table)

We explain this part of the algorithm using an example for more illustration.

For Example, Generate an equivalent SQL statement for the following joining network:

$$3 \infty 14 \infty 7 \infty 1 \infty 12$$

Part1: Collect the needed information to generate the equivalent SQL statement (see Figure (5.14))

- For each tuple (t_0, t_1, t_2…) in the joining network
    - Retrieve the table name of the tuple from tuplesInfo table (Table (5.3))
    - Append the table name in the *Tables* array
    - Append the corresponding alias name in the *Alias* array
- For each pair of neighbor tuples in the joining network
    - Retrieve the primary and the foreign column names from the *Pkfk_Table* (Table (5.6)), which connects the pair elements in primary to foreigh key

relationship, and append them in the *Relationship* array with the same order of the pair elements

- Get the share value that joins the pair elements in primary to foreign key relationship from the *ShareV* field in the *Pkfk_Table*, and append it in the *ShareV* array.

- Store the symbol (P) to use as an indication for the primary part of the relation and the symbol (F) to use as indication for the foreign part of the relation in their corresponding location in the *PF_Part* array

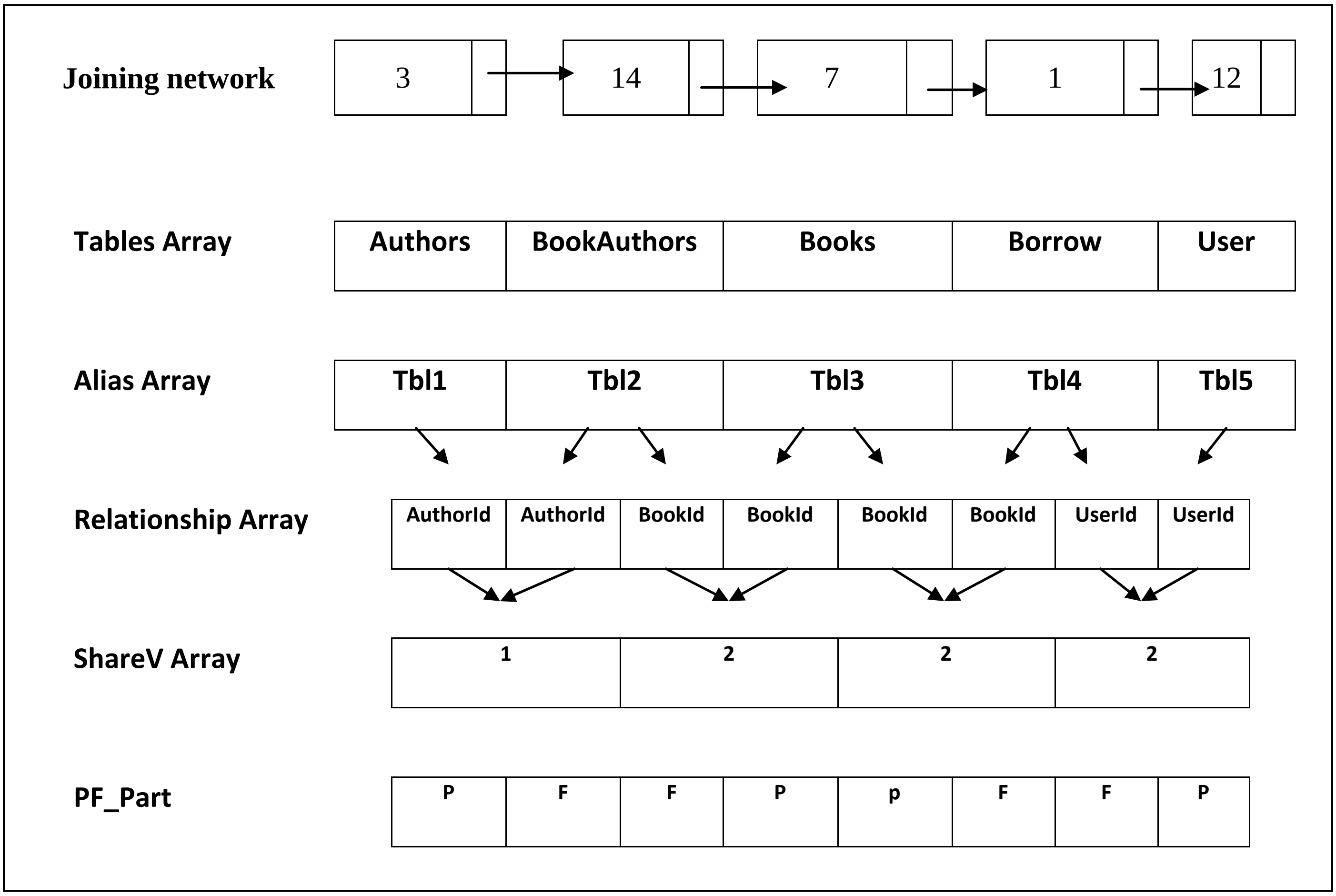


Figure 5.14: Example of the needed information for generating an equivalent SQL statement for the joining network 3 ∞ 14 ∞ 7 ∞ 1 ∞ 12

Part2: Generate the equivalent SQL statement (see Figure (5.15))

1. Generate the SELECT part of the SQL statement

    - s= “select “
    - Retrieve the columns names of the first table in the *Table* array using the database catalog
    - For each column name
        - Append the column name to the string (s) using the following expression:

          s=s + alias[0] + ‘.’ + columnName +’,’
    - For each of the remaining tables in the *Tables* array
        - Retrieve the table columns names using the database catalog
        - For each columnName
            - If the columnName is not one of the column names of the previous table //to remove the redundancy columns
                - Append the columnName to the variable (s) using the following expression :

                  //alias[i] is the alias of the current table

                  s=s + alias[i] + ‘.’ + columnName +’,’
    - Remove the postfix character (,) from the string s

See Figure (5.15), SELECT part

2. Generate the FROM part of the SQL statement

- f = “ from ”
- For each table in the Tables array
    - Append the table name and it’s corresponding alias name to the variable f

      f= f + tables[i] +’ ’ +alias[i] +‘,’
- Remove the postfix character (,) from the string f

See Figure (5.15), FROM part

3. Generate the WHERE part of the SQL statement

// Generating the inner join part of the where condition

- i=0
- k=0
- w = “ where ”
- For each neighbor pairs in the joining network list

/*Retrieve the corresponding joining columns names from the *RelationShip* array with their corresponding tables name alias from *Alias* array */

  - w = w + alias[i] + ‘.’ +relationship[k]= alias[i+1] + ‘.’ + relationship [k+1] + ”and”
  - k=k+2

```
        • i=i+1

  //Retrieving the intended record from the primary part of the relationship

    • k=0;

    • For( i=0; i<length(Tables array)-1 ; i++)

        • If (PF_PART[K]='P') then    //if the first part of the relationship  is
          primary

            • w = w + alias[i] +'.' +relationship[k]+ '=' +shareV[i] + "and
              ";

        • Else    // if the second part of the relationship  is primary

            • w = w + alias[i+1] +'.' +relationship[k+1]+ '=' +shareV[i] +
              "and ";

        • k=k+2

  //Retrieving the intended record from the foreign part of the relation

    • k=0;

    • For( i=0; i<length(Tables array)-1 ; i++)

        • If (PF_PART[K]='F') then    //if the first part of the relationship  is
          foreign

            • Retrieve the primary column name and the primary key
              value for the foreign part of the relationship using the
              corresponding table name (table[i]) and the tuple number at
              the corresponding location in the joining network list  using
              the DB catalog
        • w = w + alias[i] +’.’ + primaryColumn + ‘=’ + primaryValue
          + “and ”;

    • Else     // if the second part of the relationship  is foreign

        • Retrieve the primary column name and the primary key
          value for the foreign part of the relationship using the
          corresponding table name (table[i+1]) and the tuple number
          at the corresponding location in the joining network list
          using the DB catalog

        • w= w + alias[i+1] +’.’ + primaryColumn + ‘=’ +
          primaryValue + “and ”

        • k=k+2

    • Remove the postfix (and) from the string w
```

See Figure (5.15), WHERE part

4. Generate the complete SQL statement

```
• sql = s + f + w
```

Select Part

```
select
          tbl1.authorId, tbl1.firstName, tbl1.lastName,
          tbl1.nationality, tbl2.bookId, tbl3.title,
          tbl3.copyRightYear, tbl3.ISBN, tbl4.borrowKey,
          tbl4.userId, tbl4.return, tbl5.firstName, tbl5.lastName,
          tbl5.address, tbl5.phone
```

From Part

```
from
Authors tbl1, BookAuthors tbl2, Books tbl3, Borrow tbl4, User
tbl5
```

Where Part

```
where
tbl1.authorId= tb2.authorId and tbl2.bookId= tb3.bookId and
tbl3.bookId= tbl4.bookId and tbl4.userId = tbl5.userId   and
tbl1.authorId=1 and tbl3.bookId=2 and tbl5.userId=2 and
tbl2.bookId=2 and tbl2.authorId=1 and tbl4.borrowKey= 1
```

Figure 5.15: The different parts of the equivalent SQL statement for the joining network 3 ∞ 14 ∞ 7 ∞ 1 ∞ 12.

The following table shows the candidates networks and their equivalents SQL statements (using the SQL answer generator algorithm) of the request query:

author:nancy book:planning

Table 5.7: The candidates networks and their equivalent SQL statements of the query “author:nancy book:planning”.

Joining Network	SQL Statement
3 ∞14 ∞ 7	**select** tbl1.authorId, tbl1.firstNmae, tbl1.lastName, tbl1.nationality, tbl2.bookId, tbl3.title, tbl3.copyRightYear, tbl3.ISBN **from** Authors tbl1, BookAuthors tbl2, Books tbl3 **where** t1.authorId= t2.authorId and t2.bookId= t3.bookId and t1.authorId=1 and t3.bookId=2 and t2.bookId=2 and t2.authorId=1
3 ∞ 14 ∞ 7 ∞ 1 ∞12	**Select** t1.authorId, t1.firstName, t1.lastName, t1.nationality, t2.bookId, t3.title, t3.copyRightYear, t3.ISBN, t4.borrowKey, t4.userId, t4.return, t5.firstName, t5.lastName, t5.address, t5.phone **from** Authors t1, BookAuthors t2, Books t3, t4 Borrow, t5 User **where** t1.authorId= t2.authorId and t2.bookId= t3.bookId and t3.bookId= t4.bookId and t4.userId = t5.userId t1.authorId=1 and t3.bookId=2 and t5.userId=2 and t2.bookId=2 and t2.authorId=1 and t4.borrowKey= 1
12 ∞ 1 ∞ 7	**select** t1.userId, t1.firstName, t1.lastName, t1.address, t1.phone, t2.borrowKey, t2.bookId, t2.return, t3.title, t3.copyRightYear, t3.ISBN **from** User t1, Borrow t2, Books t3 **where** t1.userId= t2.userId and t2.bookId=t3.bookId and t1.userId= 2 and t3.bookId= 2 and t2.borrowKey=1

5.5 Characteristics of *Ssearch*

1. The ability to apply semantics for the search keyword term.
2. Provide the most relevancy results at first top-k.
3. Using the capabilities of DBMS to improve and accelerate information retrieval (such as the indexing technique, SQL statements capabilities).
4. Apply the results that satisfy AND semantics.
5. The ability to formulate query which have FOR ALL semantics implicitly.
6. Cashing the query results.
7. Stemming the search terms and the indexing terms.
8. Discard stop words (from the search terms and the indexing terms).

5.6 Summery

A *search operation* is initiated by typing the user the query syntax in the text box and then press the search button, ***Ssearch*** first parsing the query syntax by splitting the contents of the query string into different tokens , some tokens are keywords terms and others are semantic terms, ***Ssearch*** associate each semantic term to its corresponding keyword term, then retrieves all the matching records that have at least one of the user keywords, then checks if the matching record satisfy the user semantic or not, ***Ssearch*** build all the different pairs combinations of tuples (matching records) to find the shortest path for each combination, the shortest path of each combination represent a joining network, it ranks the joining networks according to their ranking score in descending order which take into a count the user semantics satisfaction, then generates the equivalent SQL statement for each joining network and finally it executes the SQL statements over original database to retrieve the user's request results.

Chapter Six: Experimental Design and Results Analysis

6.1 Introduction

As stated earlier the main objective of the proposed system is to improve the total number of relevant results at the Top-k according to user query, within acceptable overheads in time.

We design experiments in order to compare the relevance of proposed system ***Ssearch*** with ***Discover***.

This chapter presents the experiments setup, experiments procedures, experiments objectives, experiments results and analysis to test the system outlined in the previous chapter.

6.2 Experiments Setup

The experimental environment design implements the experimental systems (***Discover, Ssearch***) using Java language (jdk1.6 compiler) and Oracle 10g DB, we run the both systems on the same PC under windows XP, with Intel Core 2 Due (2.0GHZ 2MB cache) and 4 GB of RAM.

We used Al–Quds university library database for evaluation. The ER-Diagram of the experimental database is shown in Figure (6.1). We got randomly two samples of this database, the first sample contains data about 5000 books with 31297 tuples (records) and size 10 MB, while the second one contains data about 10000 books with 97282 tuples and size 29 MB, we used these two samples to study the scalability of the system.

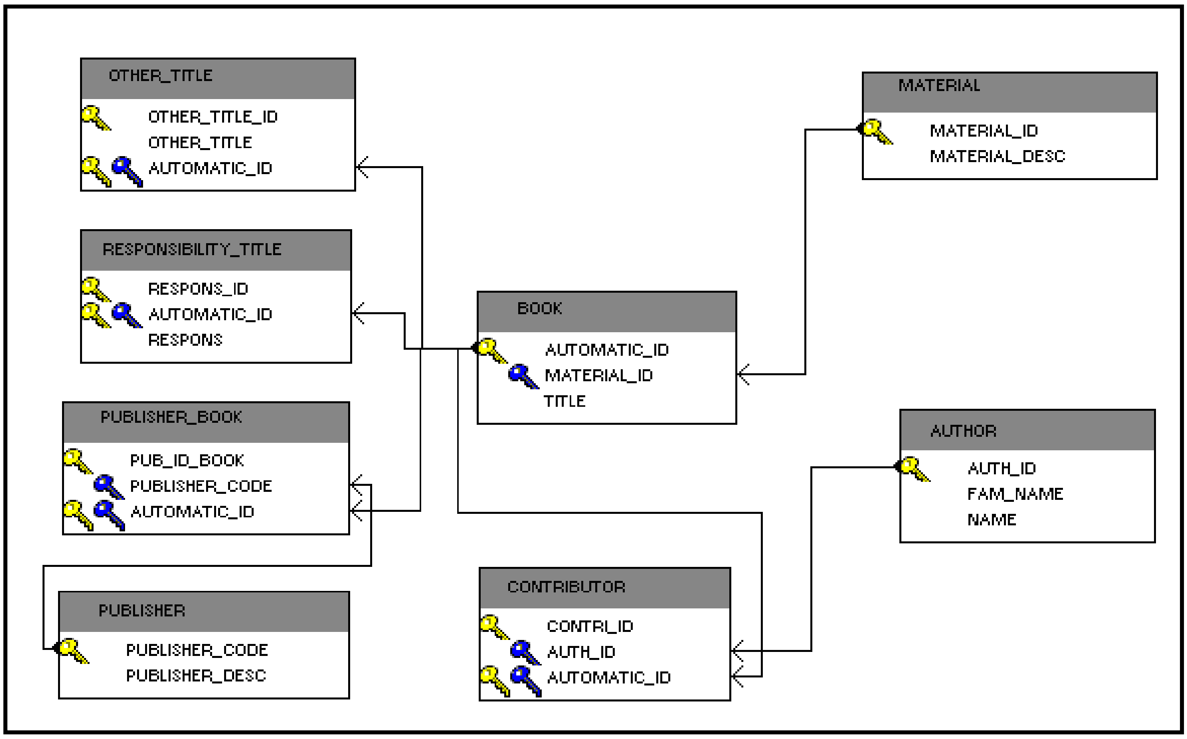


Figure 6.1: ER-Diagram of the experimental DB.

We also chose 16 different queries to be executed over the library database samples, we tried to take into account when selecting this queries, to get all the probabilities we can have from the searching operation, to get a wide view of how much the results are relevant to the user query in different situations, we changed from simple queries which contains only one keyword to more complex queries which contain more than one keyword. See Table (A.1) in the appendix.

For each selected query we identified a semantic term for each keyword in it, for example, the input query "Book:dictionary Author:Al-Kilani" retrieves all the dictionary books which are written by the author Al-kilany, where the book is the semantic term of the dictionary keyword and the author is the semantic term of AL-Kilany keyword.

6.3 Experiments Metrics

We use three metrics to evaluate the different aspects of semantic search effectiveness:

1- Number of Top-k answers that are relevant to the user query. We consider the answer is relevant, if and only if each keyword in the input query satisfies the user semantics in the answer, i.e. using the same query mentioned in the previous example (“Book:dictionary Author:Al-Kilani”), if the keyword dictionary in the answer comes from the book table and Al-Kilani keyword comes from the author table, this answer will be considered relevant for the user semantics otherwise the answer is irrelevant, for more illustration see Figure (6.2), which displays some relevant and irrelevant results for this semantic keyword query.

input search query :
“ Book:dictionary Author:Al-Kilani “

AUTHOR. AUTH_ID	AUTHOR .FAM_NAME	AUTHOR. FIRST_NAME	CONTRIBUTOR. CONTRI_ID	CONTRIBUTOR. AUTOMATIC_ID	BOOK. MATERIAL_ID	BOOK. TITLE	
9573	Al-Kilani	Taiseer	1	10973	1	An encyclopedic dictionary of games and sports (English-Arabic)	Rel.

AUTHOR. AUTH_ID	AUTHOR .FAM_NAME	AUTHOR. FIRST_NAME	CONTRIBUTOR. CONTRI_ID	CONTRIBUTOR. AUTOMATIC_ID	BOOK. MATERIAL_ID	BOOK. TITLE	
9573	Al-Kilani	Taiseer	1	12785	1	The Al-Kilani dictionary of computer and internet terminology : English – English – Arabic, with illustrations	Rel.

PUBLISHER. PUBLISHER_CODE	PUBLISHER. PUBLISHER_DISC	PUBLISHER.PUBLISHER PUB_ID_BOOK	BOOK. AUTOMATIC_ID	BOOK. MATERIAL_ID	BOOK. TITLE	
105	Librairie du Al-Kilani	1	13179	1	A dictionary of economic and financial terms : English – French – Arabic : with indexes of French and Arabic key-words	Not Rel.

PUBLISHER. PUBLISHER_CODE	PUBLISHER. PUBLISHER_DISC	PUBLISHER.PUBLISHER PUB_ID_BOOK	BOOK. AUTOMATIC_ID	BOOK. MATERIAL_ID	BOOK. TITLE	
105	Librairie du Al-Kilani	1	13316	1	Faruqi s law dictionary : English - Arabic : meanings and definitions of terms of English and American Jurisprudence (ancient and modern), forensic medicine, commerce, banking, insurance, civil aviation, diplomacy and petroleum	Not Rel.

Figure 6.2: Some relevant and irrelevant answers.

2- Precision of query answers, this metric used to find the fraction of a search output that is relevant for a particular query, precision is defined as:

$$\text{precision} = \frac{|\{\text{relevant answers}\} \cap \{\text{retrieved answers}\}|}{|\{\text{retrieved answers}\}|}$$

3- Time Overhead, which we used to find the additional time needed by the proposed system to process the user semantics (parsing the semantic query, associate the query keywords with their related semantics, find if the retrieving results satisfy the user semantics or not etc.). This metric give us an indication if the proposed system running within acceptable overhead time to encourage who builds such systems to adopt the proposed idea.

6.4 Experiments Outline

We have conducted two experiments in order to verify the proposed system:

- Experiment one, testing the relevancy of the search results.
- Experiment two, testing the scalability of the system in terms of relevancy and overhead time.

6.4.1 Experiment 1: Testing the relevancy at Top-10, Top-20, Top-30, Top-40 and Top-50

This experiment give us a detail view for the effect of the proposed system ***Ssearch*** in ranking the relevant answers by comparing it with ***Discover*** at Top-10, Top-20, Top-30, Top-40 and Top-50, which is enough to study.

According to Hedger [Hedger, 2006], where 90% of search users will click on links found in the first three pages of search results, 62% of them click on a first page result. 41% of respondents would either alter or abandon the keywords used in their queries if they could not find results on the first page. Another important note is that only 12% of respondents would follow a search past the first three pages of results, down from 22% in 2002.

The proposed system displays the first Top-10 answers in the first page and the first Top-20 answers in the first two pages and the first Top-30 answers in the first three pages and so on…

The users who will use the proposed system are the same users who are usually used the web search engines, which we expect that they will follow the same behavior as mentions in [Hedger, 2006].

6.4.1.1 Procedure of Experiment 1:

In this experiment we need to compute the total number of relevant results at each Top k in both systems for the entered keyword phrase using the following procedure:

In ***Ssearch***, after inserting the semantic query, the results will be sorted in descending order based on the ***Ssearch*** ranking score, then the total number of relevant results that satisfy the user semantics at each Top-k will be computed, where $k \in \{10,20,30,40,50\}$.

In ***Discover***, we applied the same input query as we applied in ***Ssearch*** but with no semantics, we assume the same semantics implicitly, after sorting the results in descending order based on the ***Discover*** ranking score, we calculate the total number of relevant results at each Top-k, where $k \in \{10,20,30,40,50\}$.

The query answers with a higher score are more relevance to the user semantics than the lower score.

We run this experiment over the sample of the library database that contains data about 5000 books with size 10 MB.

6.4.1.2 Part 1 of Experiment 1: Testing the relevancy at Top-10

Objectives :

In this part we studied the relation between the total number of relevant answers and the Top-10 answers for query in both systems ***Ssearch*** and ***Discover***, then compared between them according to the relevancy. See Figure (6.3).

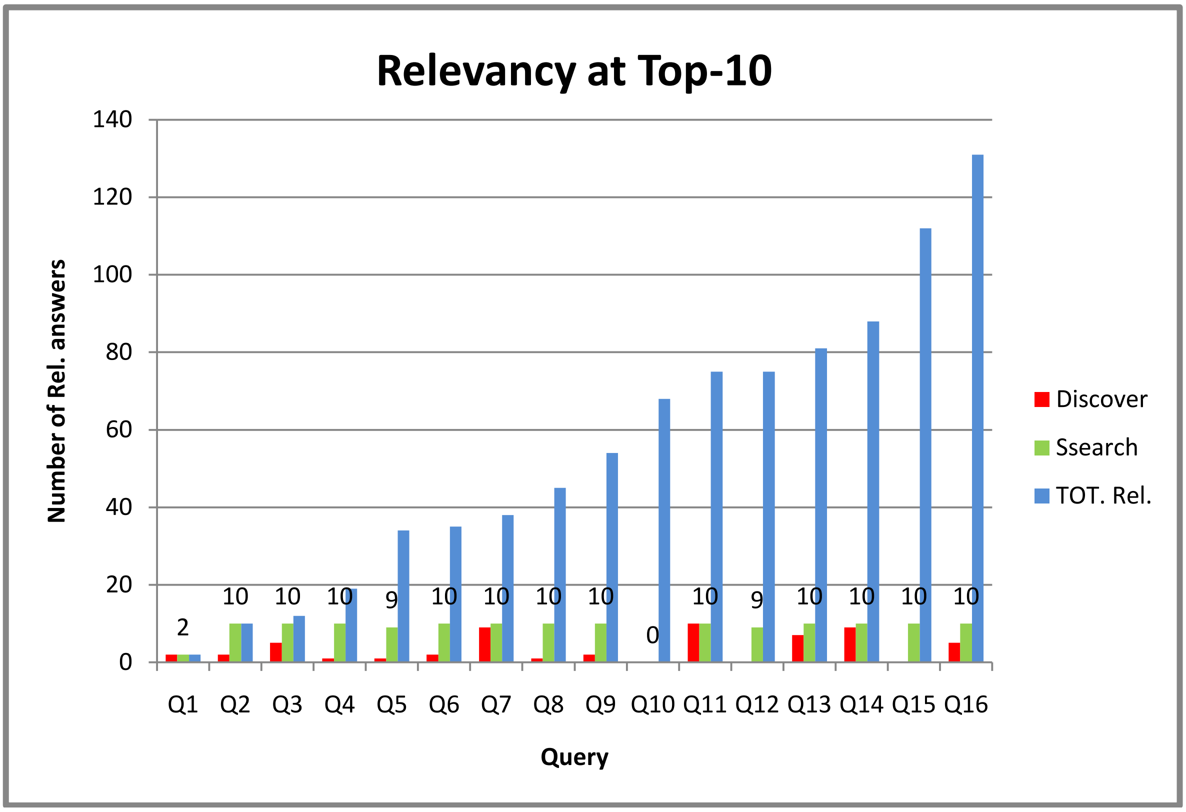


Figure 6.3: Relevancy at Top-10 (10 MB).

Discussion

We noticed in this part of the experiment that all query answers at the Top-10 in ***Ssearch*** are nearly relevant except the queries that have the total number of relevant answers less than 10, which means that our system displays all the relevant results at the Top-10, we also noticed that the number of relevant results at Top-10 in ***Ssearch*** are always greater than or equal ***Discover***. An irregular number of relevant results in ***Search*** happened only at Q10 where the number of relevant results at Top-10= 0, when we analyze the answers of this query, we found that the relevant answers of this query have a long size which leads to decrease the ranking score of such results. The longer relevant answers have smaller score in the proposed ranking algorithm than the relevant answers with shortest size.

6.4.1.3 Part 2 of Experiment 1: Testing the relevancy at Top-20

Objectives:

In this part we studied the relation between the total number of relevant answers and the Top-20 answers for query in both systems ***Ssearch*** and ***Discover***, then compare between the two systems according to the relevancy. See Figure (6.4).

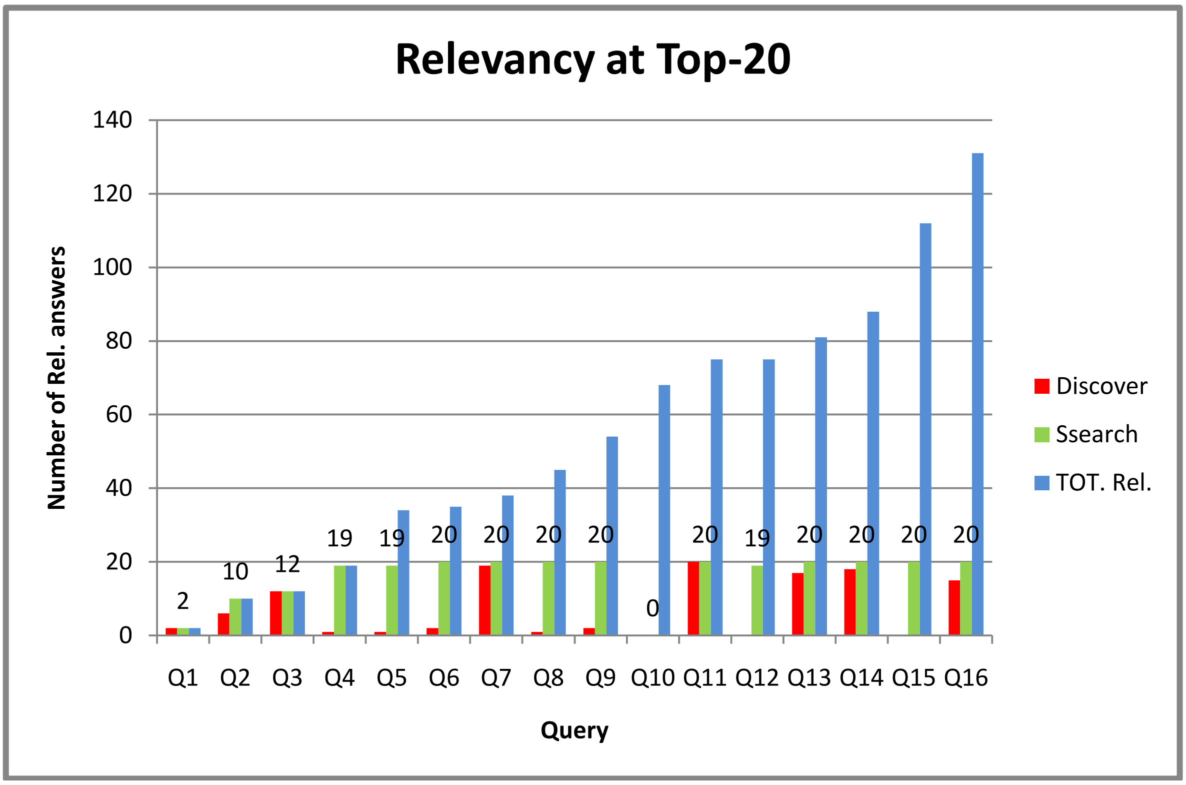


Figure 6.4: Relevancy at Top-20 (10 MB).

Discussion

We noticed in this part of the experiment that all query answers at the Top-20 in ***Ssearch*** are nearly relevant except the queries that have the total number of relevant answers less than 20, which means that our system displays all the relevant results at the Top-20, we also noticed that the number of relevant results at Top-20 in ***Ssearch*** is always greater than or equal ***Discover***. An irregular number of relevant results in ***Ssearch*** still happen at Q10 where the number of relevant results at Top-20= 0, when we analyze the answers of this query we found that the relevant answers of this query have a long size which leads to decrease the ranking score of such results. The longer relevant answers have smaller score in the proposed ranking algorithm than the relevant answers with shortest size.

6.4.1.4 Part 3 of Experiment 1: Testing the relevancy at Top-30

Objectives:

In this part we studied the relation between the total number of relevant answers and the Top-30 answers for each query in both systems ***Ssearch*** and ***Discover***, then compared between the two systems according to the relevance. See Figure (6.5).

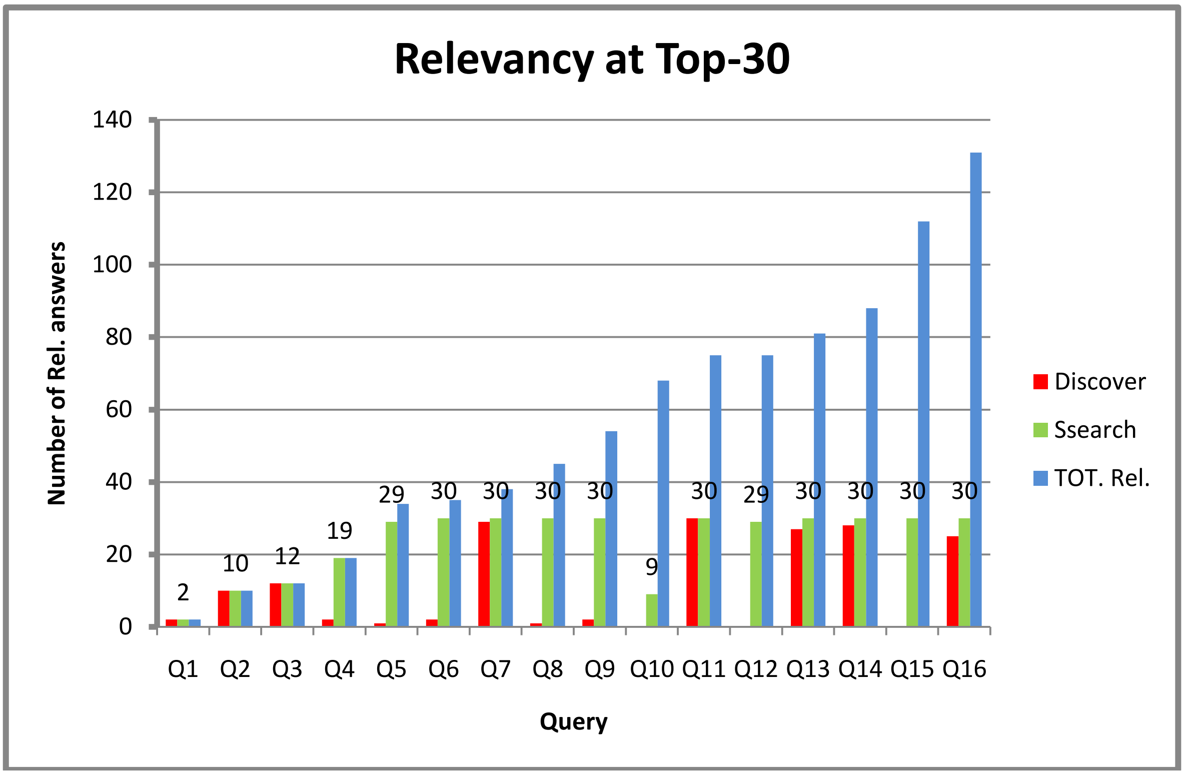


Figure 6.5: Relevancy at Top-30 (10 MB)

Discussion

We noticed in this part of the experiment that all query answers at the Top-30 in ***Ssearch*** are nearly relevant except the queries that have the total number of relevant answers less than 30, which means that our system display all the relevance results at the Top-30. We also noticed that the number of relevant results at Top-30 in ***Ssearch*** are always greater than or equal ***Discover***. The first relevant results of an irregular query (Q10) started appearing at Top-30, which display 9 relevant results from 68 relevant results while the number of relevant results in ***Discover*** still 0.

6.4.1.5 Part 4 of Experiment 1: Testing the relevancy at Top-40

Objectives:

In this part we study the relation between the total number of relevant answers and the Top-40 answers for query in both systems ***Ssearch*** and ***Discover***, then compare between the two systems according to the relevancy. See Figure (6.6).

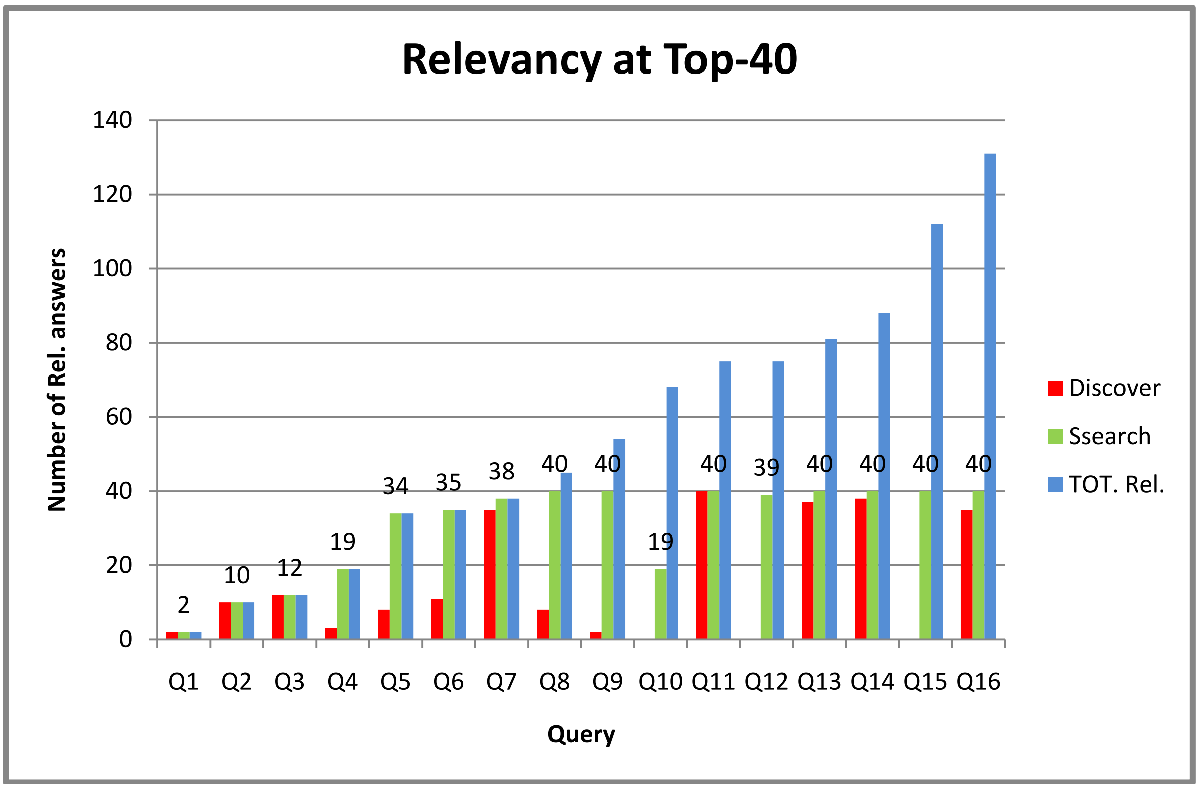


Figure 6.6: Relevancy at Top-40 (10 MB).

Discussion:

We noticed in this part of the experiment that all query answers at the Top-40 in ***Ssearch*** are nearly relevant except the queries that have the total number of relevant answers less than 40, which means that our system displays all the relevant results at the Top-40. We also noticed that the number of relevant results at Top-40 in ***Ssearch*** are always greater than or equal ***Discover***. The number of relevant results of an irregular query (Q10) at Top-40 increased by 10 which means all the results at page four are relevant while the number of relevant results in ***Discover*** still 0.

6.4.1.6 Part 5 of Experiment 1: Testing the relevancy at Top-50

Objectives:

In this part we study the relation between the total number of relevant answers and the Top-50 answers for each query in both systems ***Ssearch*** and ***Discover***, then compared between them according to the relevancy. See Figure (6.7).

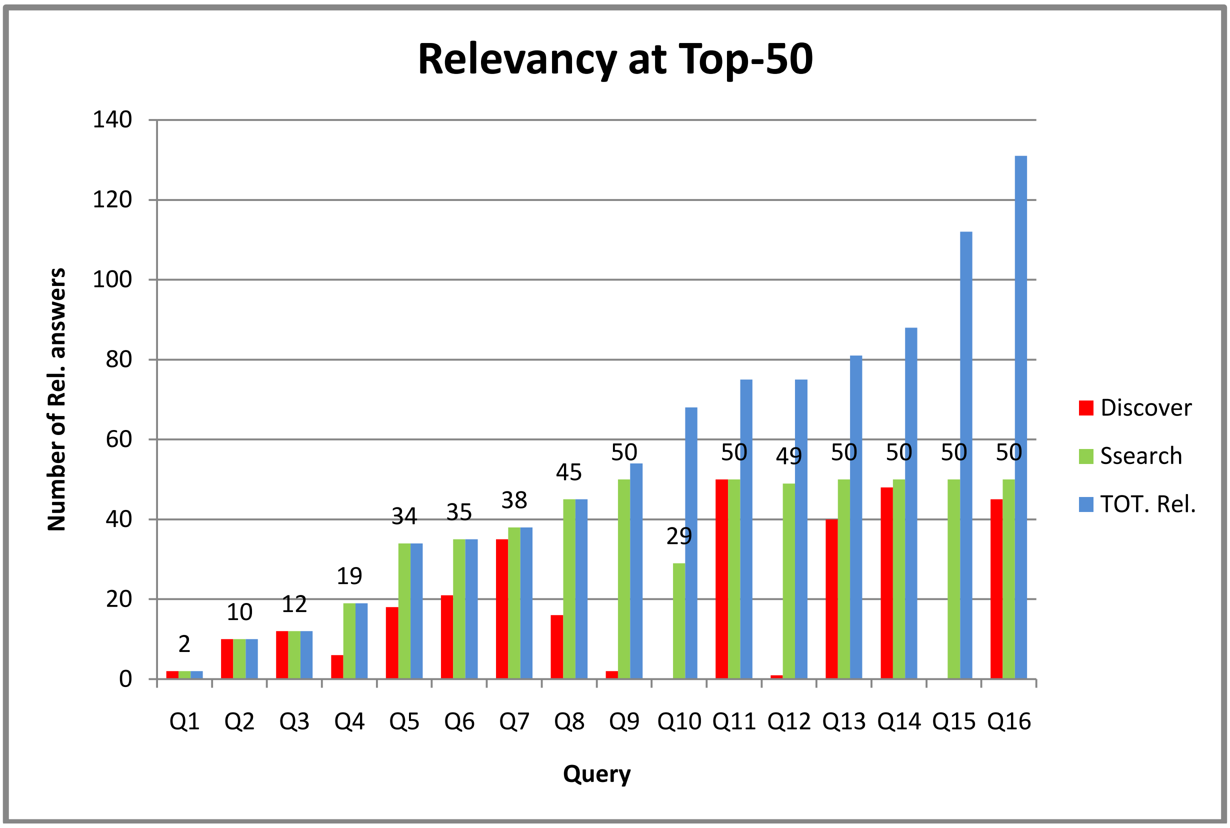


Figure 6.7: Relevancy at Top-50 (10 MB).

Discussion

We noticed in this part of the experiment that all query answers at the Top-50 in ***Ssearch*** are nearly relevant except the queries that have the total number of relevant answers less than 50, which means that our system displays all the relevant results at the Top-50. We also noticed that the number of relevant results at Top-50 in ***Ssearch*** are always greater than or equal ***Discover***. The number of relevant results of an irregular query (Q10) at Top-50 increase by 10, which means that all the results at page five also are relevant while the number of relevant results in ***Discover*** still 0.

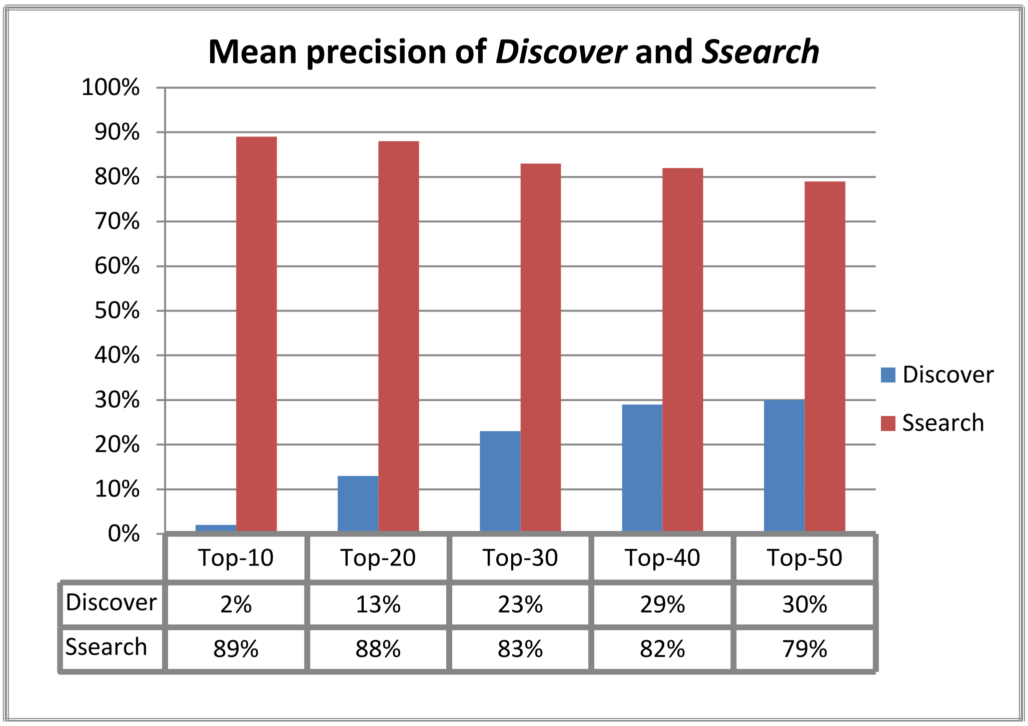
6.4.1.7 Overall Discussion of Experiment 1:

Based on all the above figures, we noticed that all the query answers at the Top-k in ***Ssearch*** are nearly relevant except the queries that have the total number of relevant answers less than k, which means that most of the results at top-k in ***Ssearch*** are relevant to a user query.

We also noticed that the number of relevant results at Top-k in ***Ssearch*** are always greater than or equal the number of relevant results at Top-k in ***Discover.***

Even by using an irregular query (Q10), the proposed system verified based on the above experiments that ***Ssearch*** gives a higher score for the relevant results than ***Discover*** in any case, the relevant results of this query started to appear at Top-30 and increasing at Top-40 and Top-50, while ***Discover*** displays the top-50 results, with no relevant results.

To get a general view of experiment one, we compare the mean precision between ***Discover*** and ***Ssearch***. See Figure (6.8)



	Top-10	Top-20	Top-30	Top-40	Top-50
Discover	2%	13%	23%	29%	30%
Ssearch	89%	88%	83%	82%	79%

Figure 6.8: Mean precision of *Discover* and *Ssearch* (10 MB).

We noticed that ***Ssearch*** have the highest precision at any top-k, the precision at the first page (Top 10) which is the most concern by all the users have a high precision 88%, while D***iscover*** have 35% .

All the results in this experiment revealed that the proposed system (semantic search engine over relational database) contributes in achieving better performance in terms of relevancy.

6.4.2 Experiment 2: Testing the scalability of the system in terms of relevancy and overhead time

To measure the scalability of the proposed system, we have conducted one experiment with two different outputs, relevancy and overhead time:

- Part1: Testing the relevancy at Top-10, Top-20, Top-30, Top-40 and Top-50.

- Part2: Testing ***Ssearch*** overheads in terms of time.

We run this experiment using a bigger sample size of the library database than that in experiment one which contains data about 10000 books and 97282 records with size 29 MB.

6.4.2.1 Part1 of Experiment 2: the relevancy at Top-10, Top-20, Top-30, Top-40 and Top-50

This part of the experiment give us a detail view for the effect of the proposed system ***Ssearch*** in ranking the relevant answers by comparing it with ***Discover*** at Top-10, Top-20, Top-30, Top-40 and Top-50 using another sample of the library database which is greater than the sample used in the first experiment.

6.4.2.1.1 Experiment Procedure of Part1:

The procedure of this part of the experiment is the same as the procedure of experiment one (section, 6.4.1.1), except the sample of the experimental database in this part is bigger in size.

6.4.2.1.2 Part 1.1 of Experiment 2: Testing the relevancy at Top-10

Objectives:

In this part we studied the relation between the total number of relevant answers and the Top-20 answers for query in both systems ***Ssearch*** and ***Discover***, then compare between the two systems according to the relevancy. See Figure (6.9).

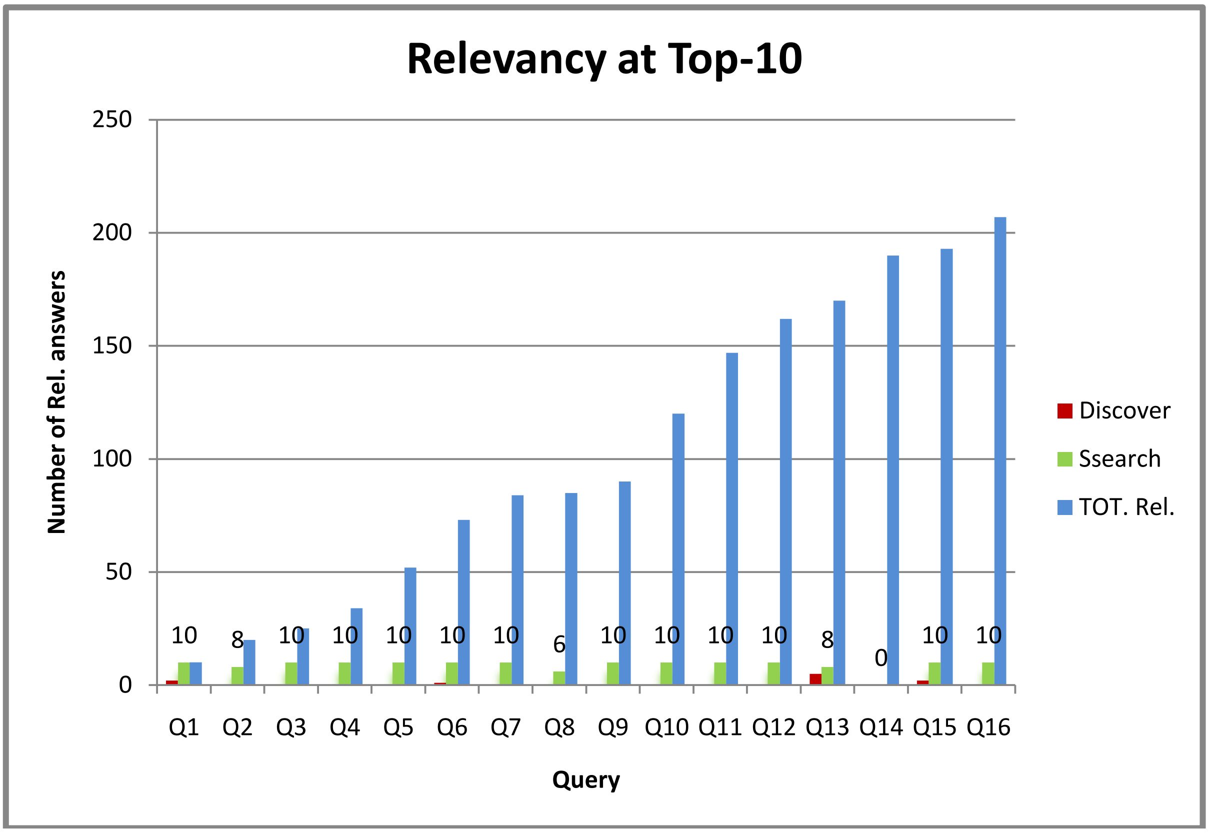


Figure 6.9: Relevancy at Top-10 (29 MB).

Discussion

We noticed in this part of the experiment that most of the query answers at the Top-10 in ***Ssearch*** are relevant while in ***Discover*** are irrelevant, except the Q14 where the number of its relevant results at top-10=0, when we analyze the answers of this query, we found that the relevant answers of this query have a long size which leads to decrease the ranking score of such results. We also noticed that the total number of relevant results at Top-10 in ***Ssearch*** is always greater than or equal ***Discover***.

6.4.2.1.3 Part 1.2 of Experiment 2: Testing the relevancy at Top-20

Objectives:

In this part we studied the relation between the total number of relevant answers and the Top-20 answers for query in both systems ***Ssearch*** and ***Discover***, then compare between the two systems according to the relevancy. See Figure (6.10).

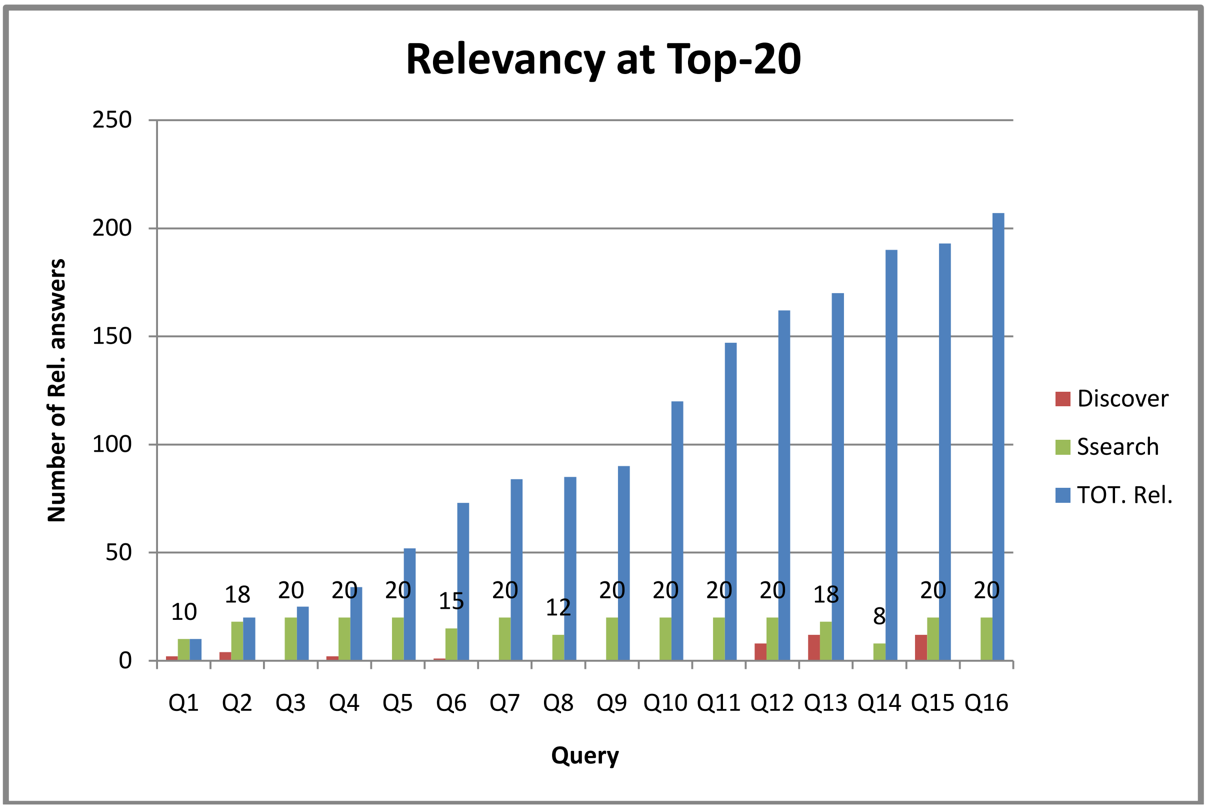


Figure 6.10: Relevancy at Top-20 (29 MB).

Discussion

We noticed in this part of the experiment that most of the query answers at the Top-20 in ***Ssearch*** are relevant while in ***Discover*** are irrelevant, the relevant answers of Q14 started appearing in the second page while ***Discover*** still have no relevant results. We also noticed that the total number of relevant results at Top-20 in ***Ssearch*** is always greater than the total number of relevant results in ***Discover.***

6.4.2.1.4 Part 1.3 of Experiment 2: Testing the relevancy at Top-30

Objectives:

In this part we studied the relation between the total number of relevant answers and the Top-30 answers for query in both systems ***Ssearch*** and ***Discover***, then compare between the two systems according to the relevancy. See Figure (6.11).

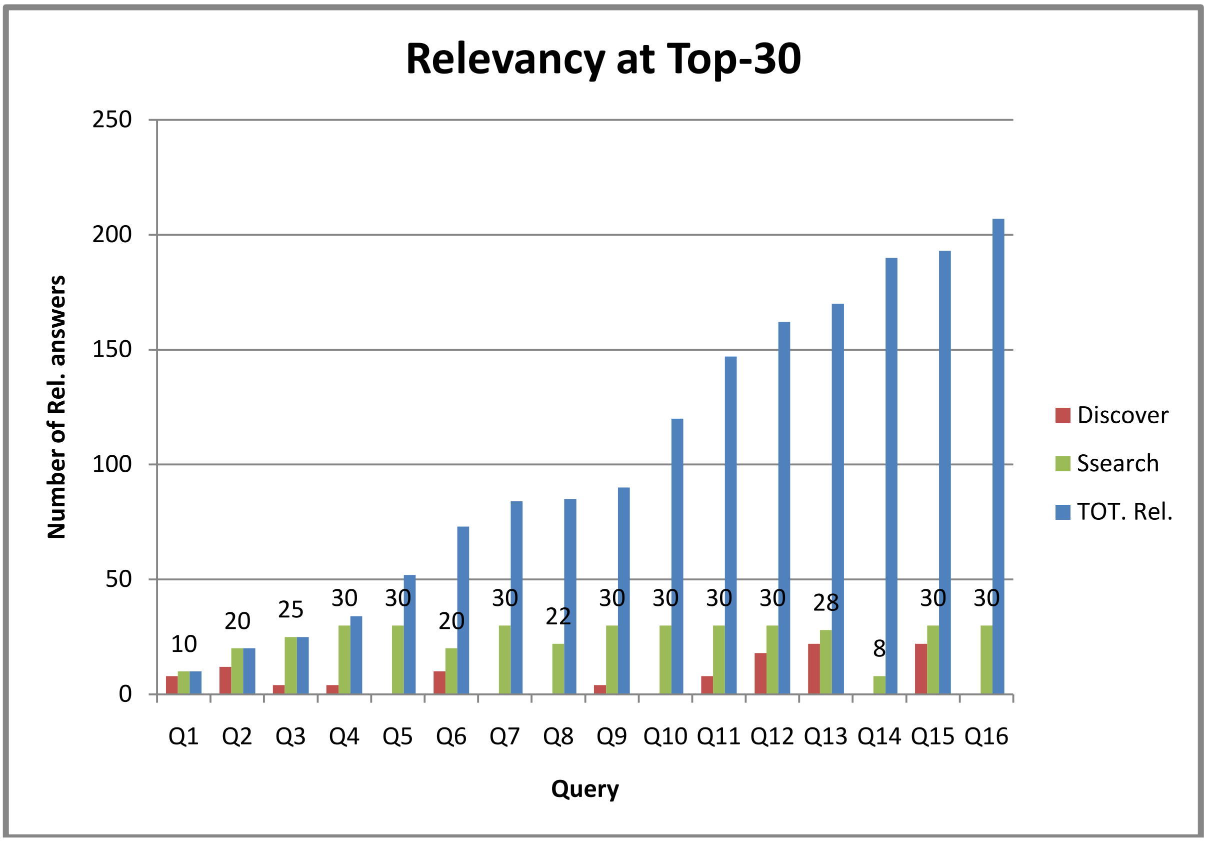


Figure 6.11: Relevancy at Top-30 (29 MB)

Discussion

We noticed in this part of the experiment that most of the query answers at the Top-30 in ***Ssearch*** are relevant. We also noticed that the total number of relevant results at Top-30 in ***Ssearch*** is greater than the total number of relevant results in ***Discover***.

6.4.2.1.5 Part 1.4 of Experiment 2: Testing the relevancy at Top-40

Objectives:

In this part we studied the relation between the total number of relevant answers and the Top-40 answers for query in both systems ***Ssearch*** and ***Discover***, then compare between the two systems according to the relevancy. See Figure (6.12).

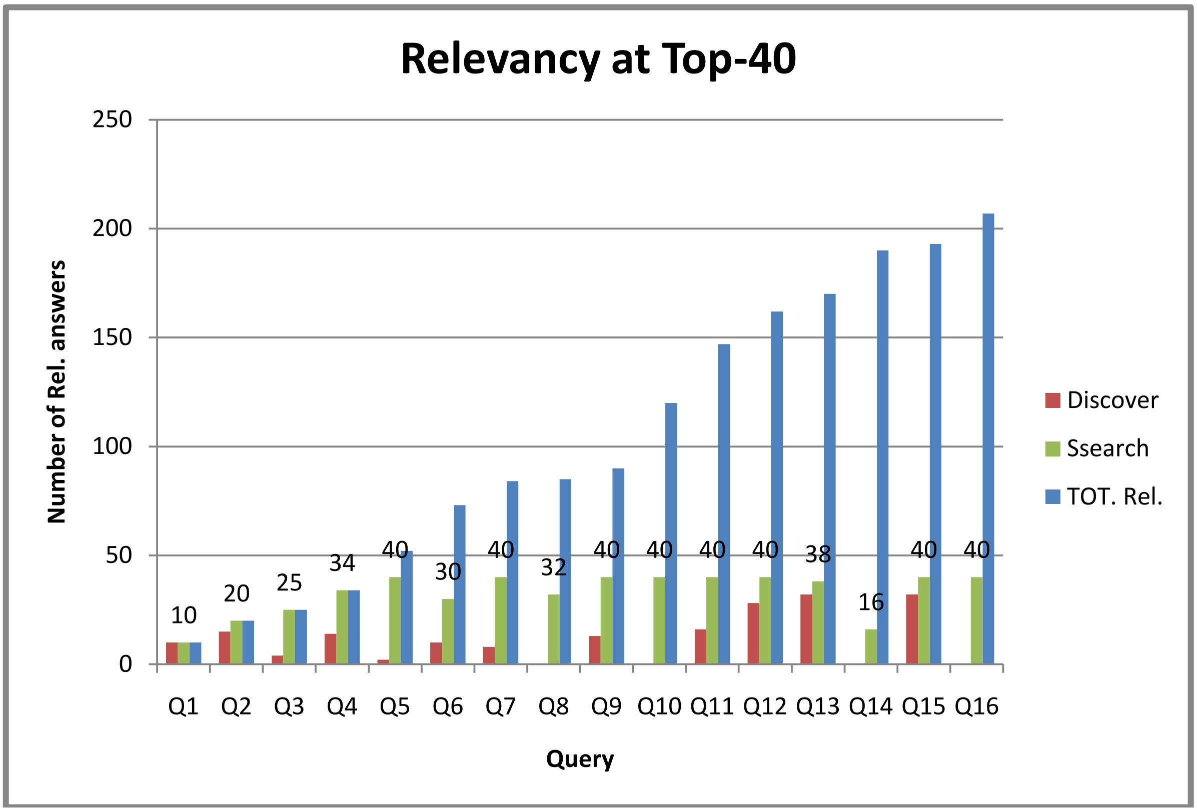


Figure 6.12: Relevancy at Top-40 (29 MB).

Discussion

We noticed in this part of the experiment that most of the query answers at the Top-40 in ***Ssearch*** are relevant. We also noticed that the total number of relevant results at Top-40 in ***Ssearch*** is greater than the total number of relevant results in ***Discover***.

6.4.2.1.6 Part 1.5 of Experiment 2: Testing the relevancy at Top-50

Objectives:

In this part we studied the relation between the total number of relevant answers and the Top-50 answers for query in both systems ***Ssearch*** and ***Discover***, then compare between the two systems according to the relevancy. See Figure (6.13).

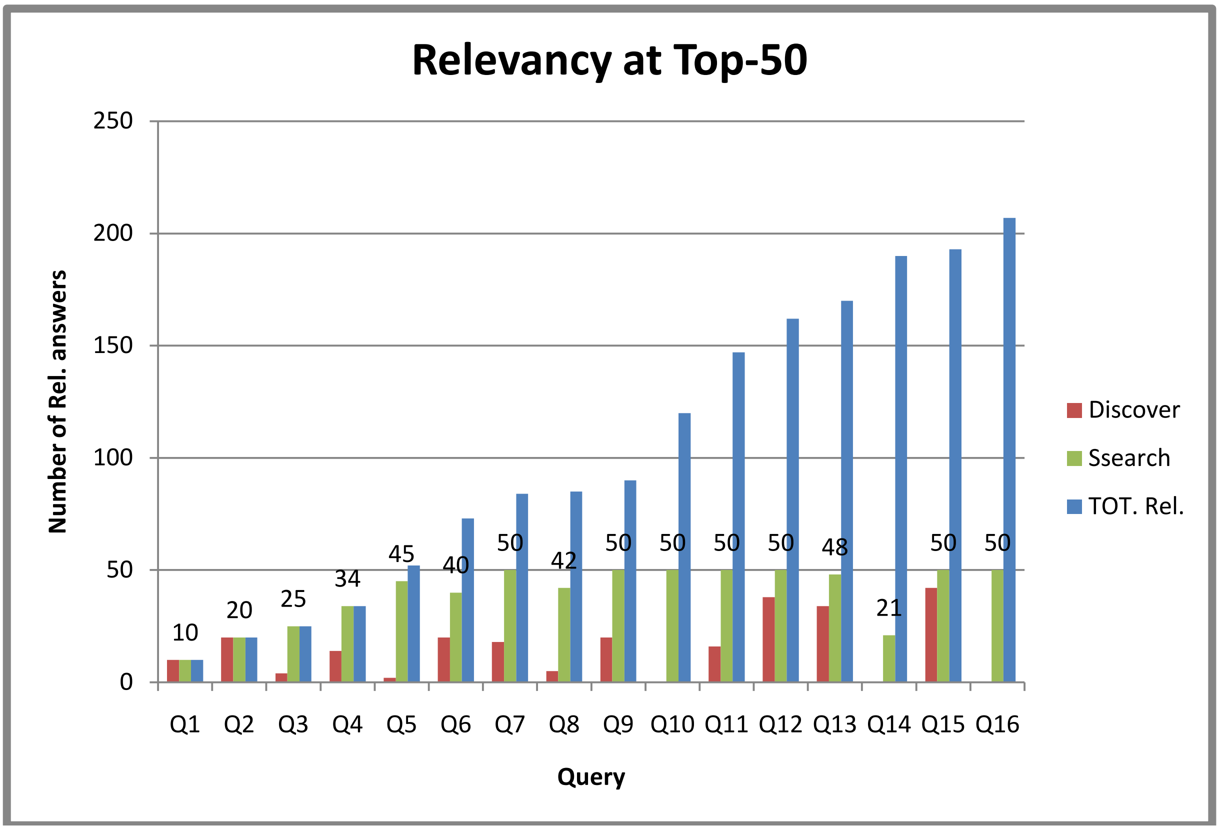


Figure 6.13: Relevancy at Top-50 (29 MB).

Discussion

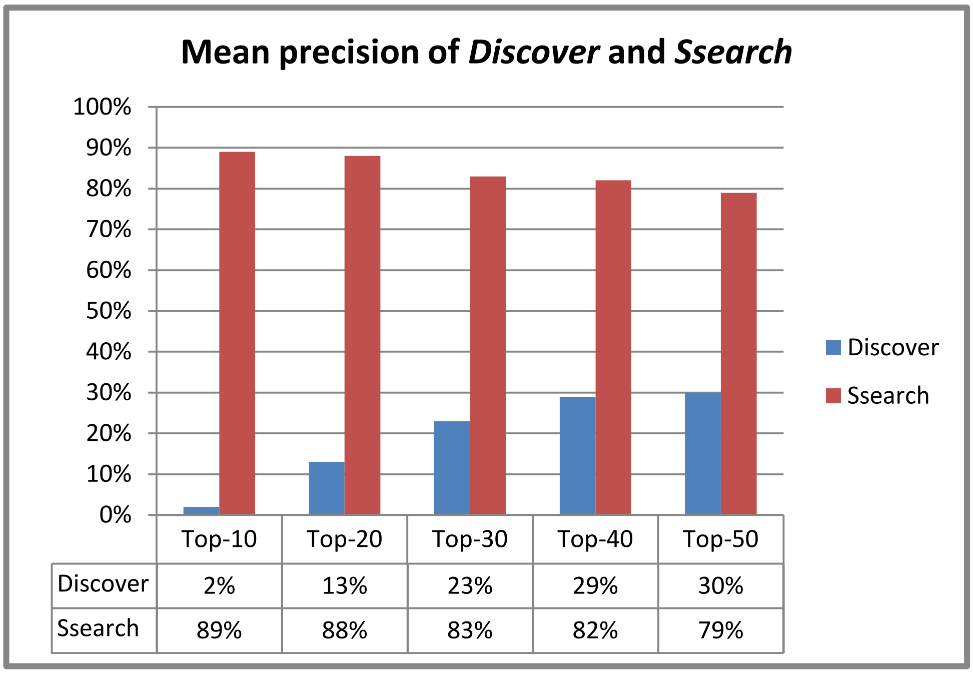
We noticed in this part of the experiment that most of the query answers at the Top-50 in ***Ssearch*** are relevant. We also noticed that the total number of relevant results at Top-50 in ***Ssearch*** is greater than or equal the total number of relevant results in ***Discover***.

6.4.2.1.7 Overall Discussion of Experiment 2 Part1:

Based on all the above figures in this part of the experiment, we noticed that all the query answers at the Top-k in ***Ssearch*** are nearly relevant except the queries that have the total number of relevant answers less than k.

We also noticed that the total number of relevant results at Top-k in ***Ssearch*** is always greater than or equal the number of relevant results at Top-k in ***Discover***.

To get a general view of this part of the experiment, we compare the mean precision between ***Discover*** and ***Ssearch***. See Figure (6.14)



	Top-10	Top-20	Top-30	Top-40	Top-50
Discover	2%	13%	23%	29%	30%
Ssearch	89%	88%	83%	82%	79%

Figure 6.14: Mean precision of Discover and Ssearch (29 MB).

We noticed when compare the mean precision between ***Discover*** and ***Ssearch***, that ***Ssearch*** have the highest precision at any top-k, the precision at the first page (Top 10) which is the most concern by all the users have a high precision 89%, while Discover have 2% .

All the results in this experiment revealed that the proposed system (semantic search engine over relational database) contributes in achieving better performance in terms of relevancy. Also these results give us an indication that the proposed system is scalable in terms of relevancy.

6.4.2.2 Part 2 of Experiment 2: Testing *Ssearch* overheads in terms of time

As shown in the previous part of the experiment, it was obvious that the proposed idea added a significant improvement in terms of relevancy, but this is not enough to take a decision that this idea is the better choice for building such system, because the overheads of building such system may affect the acceptable response time, which is one of the most sensitive issues in the world of data retrieval.

Response time means the time between submitting the search query and retrieving the first batch of results (first Top-10).

The overheads induced by ***Ssearch*** are the time required to process all the operations related to semantics (the proposed idea).

The following part of the experiment will give us a general estimation for the additional needed time to run such a system.

6.4.2.1 Experiment Procedure of part 2:

To compute ***Ssearch*** overheads in terms of time, we followed the following procedure:

For each query in the experimental query set. See Table (A.1).

Step1: run the query on both systems (***Ssearch***, ***Discover***) and then compute the ***Ssearch*** overheads using the following formula:

Response Time(***Ssearch***)- Response Time(***Discover***)

Step2: repeat step1, 100 times for each query to get more accurate results by computing the average overheads per query.

Overheads time included the following processes:

1- Parsing the semantic query

The time needed to distinguish between the user semantics and users keywords and associate each input keyword with its input semantic.

2- Retrieving the matching keyword

The time needed to verify if the semantic of the retrieved keywords (from database) match the user keywords semantics or not.

3- Ranking the answers

The time needed to compute the percentage of input keywords that satisfies the user semantics in the answer (this percentage used to compute the ranking score of the answer).

We used the same environment for running the both systems (***Ssearch, Discover***)

6.4.2.2 Experiment Objectives of Part 2:

In this part of the experiment we measured the time overheads due to the use of the semantic characteristic in the input search query. Table (6.1) shows the mean overheads taken by each query.

Table 6.1: Semantics Overheads per query.

Query No.	*Discover* Response time (sec)	*Ssearch* Response time (sec)	Overhead (sec)	Overhead Fraction (sec)
1	252.6798	252.7026	0.022808	9.02561E-05
2	324.85970	324.8871	0.027395	8.43217E-05
3	469.2625	469.2962	0.033700	7.18097E-05
4	758.0394	758.0972	0.057800	7.62435E-05
5	758.0433	758.1012	0.057900	7.6375E-05
6	1119.0105	1119.0816	0.071100	6.35342E-05
7	1155.1077	1155.1811	0.073400	6.35398E-05
8	1371.6903	1371.771	0.080700	5.88291E-05
9	1443.8846	1443.9671	0.082500	5.71343E-05
10	1443.8846	1443.968	0.083400	5.77575E-05
11	1588.273	1588.3631	0.090100	5.67251E-05
12	1696.5644	1696.6565	0.092100	5.42832E-05
13	2129.7297	2129.8269	0.097200	4.56375E-05
14	2887.7691	2887.8758	0.106700	3.69476E-05
15	3068.2547	3068.3651	0.110400	3.59801E-05
16	3104.3518	3104.4631	0.111300	3.58516E-05
Avg.	1473.2135	1473.2864	0.0729	4.94812E-05

During this part of experiment, we noticed that the overhead time depends on the total number of matching keywords per query. See Figure 6.15.

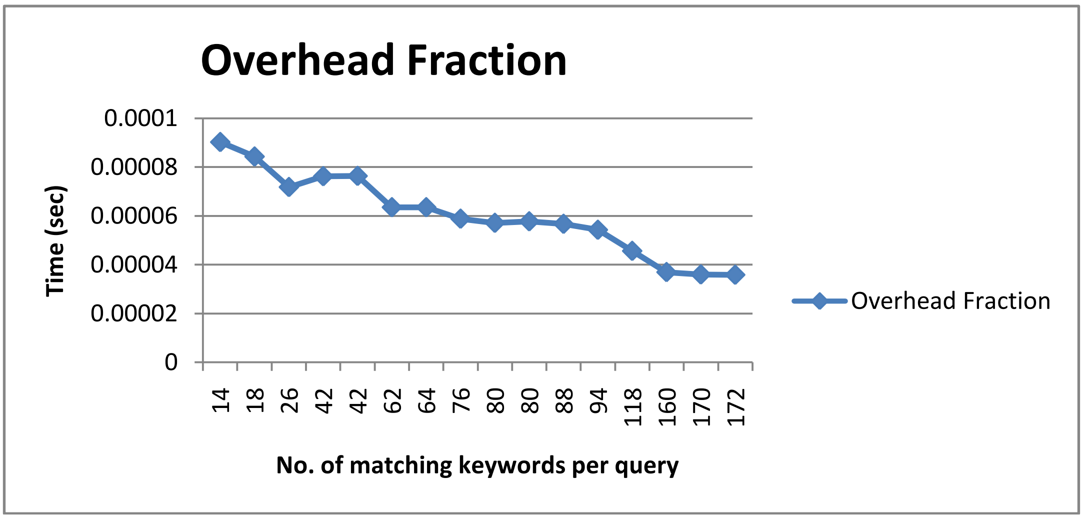


Figure 6.15: The relation between number of matching keywords and the overhead fraction per query.

Matching keyword is the keyword in the database that matches one of the input query keywords.

6.4.2.3 Experiment Discussion of part 2:

As shown in Table (6.1), the maximum overhead time for the experimental query equal 0.11 sec and the average overhead time equal 0.07 sec, which is 4.94E-5 of the total response time of ***Ssearch***, that means that ***Ssearch*** add a very tiny fraction of time to apply the proposed idea .

Also, Figure (6.15) shows that the overhead fraction depends on the total number of the keywords in the database that match one of the user keywords after stemming, if the number of matching keywords increase the overhead fraction decrease accordingly (because if the number of matching keywords increase, the total number of possible candidate networks increase accordingly, so the number of times needed to search the data graph will also increase).

This part of the experiment will give the developers of such systems an idea about an approximate overhead time needed for processing such semantics in the search query.

6.4.3 Overall Summary

In this chapter, we have studied the effect of adding semantic information (table name, field name) to the keywords in the search query on the relevancy of the returned answers. Also we have studied the scalability of the proposed system in terms of relevancy and overhead time.

We have conducted two experiments, experiment one studied the relevancy of the query answers at Top-10, Top-20, Top-30, Top-40 and Top-50, while the second one studied the accuracy and scalability of the proposed system.

The initial results, indicate a significance improvement on the returned results relevancy when the search is conducted using the system presented in this research compared with the previous system (***Discover***). Also the results indicated that the system is scalable and can retrieve relevant results at top-k from a larger database.

In the first experiment, we tested the relevancy of query answers by computing the total number of relevant answers at Top-10, Top-20, Top-30, Top-40 and Top-50 using ***Ssearch*** and compared them individually with the total number of relevant answers using ***Discover*** at Top-10, Top-20, Top-30, Top-40 and Top-50. We run this experiment over a sample of Al-Quds University Library database which contains data about 5000 books. The results of this experiment showed that all the query answers in ***Ssearch*** at the Top-k are nearly relevant except the queries that have the total number of relevance answers less than k, which mean that the proposed system can display all the relevance answers at the Top-k. Also, we have noticed that the number of relevant answers at Top-k in ***Ssearch*** are always greater than or equal the number of relevant answers at Top-k in ***Discover***.

In the part1 of the second experiment, we tested also the relevancy of query answers by computing the total number of relevant answers at Top-10, Top-20, Top-30, Top-40 and Top-50 using ***Ssearch*** and compared them individually with the total number of relevant answers using ***Discover*** at Top-10, Top-20, Top-30, Top-40 and Top-50. But this time we run it over a larger sample of Al-Quds University Library database which contains data about 10000 books. The results of this experiment showed that all the query answers in ***Ssearch*** at the Top-k are nearly relevant except the queries that have the total number of relevance answers less than k, which means that the proposed system is scalable and can display the most relevant answers at top-k from a larger database.

In the part two of the second experiment, we measured the time overheads due to the use of the semantic characteristic in the input search query. The results showed that the maximum overhead time of the experimental queries equal 0.11 sec and the average overhead time equal 0.073 sec, which is 4.94E-5 of the total response time of *Ssearch*. Also, we have noticed that the overheads time depends on the total number of the keywords in the database that match one of the user keywords after stemming, which means that when the total number of retrieving keywords increases the total time for processing the semantics operations will increase accordingly.

The performance of our proposed system is relevant to our achieved results, which adds a significant improvement in terms of relevancy with acceptable overhead time, but it seems that there is a delay in response time that needs to be looked in, and this is to be investigated in future plan as it is not part of this work.

Chapter 7: Conclusion and Future Work

7.1 Conclusion

Keyword search allows non-expert users to retrieve information from relational databases with much more flexibilities. The user retrieves information without requiring to know the schema of the database, SQL or some QBE-like interface, and the roles of the various entities and terms used in the query.

Increasing amount of data stored in the database leads to increase the amount of irrelevant results returned from the query at top-k, which have a completely different meaning than what the user really means. The improvement of the relevancy of the returned results at top-k was the main motivation to do this work.

In this thesis we proposed a simple query language, suitable for a naive user, which contains a list of pairs (semantic term and keyword term), which help the user to be more specific in expressing his needs when formulating the query. Also we proposed a new ranking method which assigns score to each query result by considering two factors, user semantics and the size for the given result. Finally, we describe the algorithms needed to build such systems.

A given keyword query in ***Ssearch*** is processed in four steps. (1) The system generates all answers (candidate networks) for the query. (2) The system computes a ranking score for each answer and ranks. (3) The system generates the equivalent SQL statement for each answer (candidate network). (3) Finally, answers are returned with semantics after execution the corresponding SQL statement over the database.

We have conducted two experiments, experiment one studied the relevancy of the query answers at Top-10, Top-20, Top-30, Top-40 and Top-50, while the second one studied the scalability of the proposed system in terms of relevancy and overhead time.

The initial results, indicate a significance improvement on the returned results relevancy when the search is conducted using the system presented in this research compared with the previous system (Discover). Also the results indicated that the system is scalable and can retrieve relevant results at top-k from a larger database.

In the first experiment, we tested the relevancy of query answers by computing the total number of relevant answers at Top-10, Top-20, Top-30, Top-40 and Top-50 using ***Ssearch*** and compared them individually with the total number of relevant answers using ***Discover*** at Top-10, Top-20, Top-30, Top-40 and Top-50. We run this experiment over a sample of Al-Quds University Library database which contains data about 5000 books. The results of this experiment showed that all the query answers in ***Ssearch*** at the Top-k are nearly relevant except the queries that have the total number of relevance answers less than k, which mean that the proposed system can display all the relevance answers at the Top-k. Also, we have noticed that the number of relevant answers at Top-k in ***Ssearch*** are always greater than or equal the number of relevant answers at Top-k in ***Discover***.

In the part1 of the second experiment, we tested also the relevancy of query answers by computing the total number of relevant answers at Top-10, Top-20, Top-30, Top-40 and Top-50 using ***Ssearch*** and compared them individually with the total number of relevant answers using Discover at Top-10, Top-20, Top-30, Top-40 and Top-50. But this time we run it over a larger sample of Al-Quds University Library database which contains data about 10000 books. The results of this experiment showed that all the query answers in ***Ssearch*** at the Top-k are nearly relevant except the queries that have the total number of relevance answers less than k, which means that the proposed system is scalable in terms of relevancy.

In the part two of the second experiment, we measured the time overheads due to the use of the semantic characteristic in the input search query. The results showed that the average overhead time equal 0.073 sec, which is 4.94E-7 of the total response time of ***Ssearch***, which means that ***Ssearch*** add a very tiny fraction of time to apply the proposed idea .

Consequently, The performance of our proposed system is relevant to our achieved results, which adds a significant improvement in terms of relevancy with acceptable overhead time, but it seems that there is a delay in response time that needs to be looked in, and this is to be investigated in future plan as it is not part of this work.

7.2 Future Work

As future work, ***Ssearch*** system applied semantics for the keyword query over single database, we can extend this work to applied semantics for the keyword query over multiple databases. Also we try to improve the execution time needed to generate all the candidate networks by using the characteristics of parallel programming and distributed systems and by finding a heuristic method that help the system to expect the disconnected pairs and the pairs that have a path length greater than the total number of tables in the database (***Discover*** and ***Ssearch*** ignore the results that have a path length greater than the total number of tables in the database) to decrease the total number of needed pairs to explore their connection paths in the data graph.

References

[1] Hoare, C. A. R. Quicksort. Computer Journal 5 (1): 10-15, 1962.

[2] Julie Beth Lovins. Development of a stemming algorithm. Mechanical Translation and Computational Linguistics 11:22–31, 1968.

[3] E.F. Codd. A Relational Model of Data for Large Shared Data Banks. Communications of the ACM 13 (No. 6): 377–387, 1970.

[4] F. Benjamin Zhan, Charle E. Noon. Shortest Path Algorithms: An Evaluation Using Real Road Networks. Transportation Science 32(1): 65-73, 1998.

[5] U. Masermann, G. Vossen. Schema Independent Database Querying (on and off the Web). In Proc. Of IDEAS, 2000.

[6] U.Masermann, G.Vossen. Design and Implementation of a Novel Approach to Keyword Searching in Relational Databases. In ADBISDASFAA Symposium, 2000.

[7] S. Base, A. Van Gelder. Computer Algorithms Introduction to Design & Analysis. Third Edition, Addison Wesley Longman 2000.

[8] A. Hulgeri, G. Bhalotia, C. Nakhe, S.Chakrabarti, S. Sudarshan. Keyword Search in Databases. In CiteSeer, 2001.

[9] V. Hristidis, Y. Papakonstantinou. DISCOVER: Keyword Search in Relational Databases. In VLDB, 2002.

[10] B. Aditya, G. Bhalotia, S. Chakrabarti, A. Hulgeri, C. Nakhe, and P. Sudarshan. BANKS: Browsing and Keyword Searching in Relational Databases. In VLDB, 2002.

[11] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A System for Keyword-Based Search over Relational Databases. In ICDE, 2002.

[12] S. Cohen, J. Mamou, Y. Kanza, and Y. Sagiv. XSEarch: A Semantic Search Engine for XML. In VLDB, 2003.

[13] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-style keyword search over relational databases. In HDMS, 2003.

[14] V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword proximity search on XML graphs. In ICDE, 2003.

[15] Andrey Balmin, Vagelis Hristidis, and Yannis Papakonstantinou. ObjectRank: Authority-Based Keyword Search in Databases. In Proc. of VLDB Conf., 2004.

[16] Jim Hedger, iProspect user search engine behavior Study. A Survey in www.iprospect.com, 2006.

[17] Y. Luo, X. Lin, W. Wang, and X. Zhou. SPARK: Top-k keyword query in relational databases. In Proceedings of SIGMOD, 2007.

[18] B.Yu, G.Li, K.Sollins, A.Tung, Effective Keyword-based Selection of Relational Databases. In SIGMOD, 2007.

[19] Q. Vu, B. Ooi, D. Papadies, A. Tung, A graph method for keyword-based selection of the top-k databases. In SIGMOD, 2008.

List of Appendixes

Appendix A: Experimental Query Set

No.	Query
1	Fam_name: Wolf Name: Stanley
2	Publisher: Navigation
4	Book: Agriculture Author: Boatfield
5	Other_title: Telephone Book: Networks
6	Book: Netscape Other_title: Exploring
7	Respons: Lindley Book: Image
8	Book: Assembly Publisher: Wiley
9	Book: Dictionary Author: Al-khatib
10	Book: Internet Author: Kiley Material: Textual
11	Book: Nutrition Book: Diet
12	Book: Yoga Publisher:Unwin
13	Title:Compiler Respons:Aho
14	Book : Plastics
15	Book: Medicine fam_name:Wilkinson
16	Title:Radio Respons:Ian

Appendix B: Term Index

AND semantic: Means that all the terms the user specifies in the search query must appear in the contents of the search answers.

Candidate network: Is a set of records which are connected by primary to foreign key relationship, which contains at least two keywords.

Data graph: Is graph where the nodes represent the records of the database tables and edges represent the primary to foreign key relationship between records.

Database: Any collection of data: part numbers, product codes, customer information, etc. It usually refers to data stored on a computer.

Dense graph: A *graph* in which the number of *edges* is much bigger than the possible number of vertices.

ER-diagram: An entity-relationship diagram is a specialized graphic that illustrates the interrelationships between entities in a database, entities refers to the tables of the database, the relationships between tables refers to the primary to foreign key relationships.

Foreign Key: A field in a relational table that matches the primary key column of another table.

Irrelevant answer: Answers that not satisfy the user search query.

Keyword: word used in a search query.

Keyword search: A type of search that looks for matching documents or records that contain one or more words specified by the user.

Master index: Is the database that contains all the information needed to retrieve the user request answers in efficient way.

Offline operations: Are all the process needed to build and update the master index database, which collects, parses, and stores data to facilitate fast and accurate information retrieval.

Online operations: Are all the operations needed to process the user request.

OR semantic: Means that at least one of the terms the user specifies in the search query must appear in the documents.

Primary key: A unique identifier, often an integer that labels a certain row in a table of a relational database.

Query: Queries are the primary mechanism for retrieving information from a database and consist of questions presented to the database in a predefined format.

Ranking score: It is an indicator which measures how well a particular answer is relevant to the user query. Answers with high ranking scores are more relevant than the answers with low ranking scores.

Record (tuple): A database record is a row of data in a database table consisting of a single value from each column of data in the table. The data in the columns in a table are all of the same type of data, whereas the rows represent a given instance.

Relational Database: A database that stores data in a structure consisting of one or more tables of rows and columns, which may be interconnected. A row corresponds to a record (tuple); columns correspond to attributes (fields) in the record.

Relationship: Is a link between two tables (i.e, relations). Relationships make it possible to find data in one table that pertains to a specific record in another table.

Relevant answers: Answers that satisfy the user search query.

Schema information's: also call a metadata which have information such as table names, columns name, it defines the tables the fields in each table, and the relationships between fields and tables.

Sparse graph: A *graph* in which the number of *edges* is much less than the possible number of vertices.