International Journal of Engineering & Technology, 8 (1.6) (2019) 105-111

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Graph-Based Technique for Searching Structured Databases

Mohammad Hassan

Computer Science Department, Zarqa University - Jordan
**Corresponding author E-mail: mohdzita@zu.edu.jo*

Abstract

This paper presents a graph-based technique for searching structured or relational databases using keyword queries in a similar wayto search text files using search engines. Our approach depends on identifying first the data within a database that are most likely to provide the useful result to the raised query and then search only the identified data. The proposed search algorithm uses an undirected weighted graph data structure for implementing the search process. To construct the graph, we introduced a modified function for computing edge weights which measure the connections among vertices in the graph. Experiments and the prototype implementation on real datasets prove that the proposed model is feasible and supports searching relational databases using keyword queries.

Keywords**:** *search, Relational Databases, Information Retrieval, Graph-Structured Data***.**

1. Introduction

Relational databases hold a massive amount of information, but accessing and exploring this information using keyword search is not enabled. Accessing relational database restricted to experts due to the difficulty of understanding relational query languages, the schema of the database, and the roles of the different terms and entities in the query [13], [14]. These requirements are unfriendly for non-technical users.

On the other hand, keyword-based query search commonly used over the web, also known as Information Retrieval (IR) search, is the most popular and convenient form of asearch for users. Using a friendly interface, users provide keywords relevant to their query, and the search engine returns web pages ranked based upon their relevance to the keywords [7]. Such a simplistic model of query search does not require the user to have any knowledge of the database schema or use any high-level structured programming language. Implementing models that bridging the divide between structured relational databases and keyword search, therefore, become a major area of research and has been paid extensive attention.

In the meanwhile, Graph-based solutions are considered one of the most important approaches for solving the keyword search over structured and relational databases [2], [22], [12]. Such solutions do not depend on the specific schema. Hence they can be used by any data model that can be modeled as a graph.

In this paper, we use a graph-based approach to solve the problem of keyword search over structured and relational databases. The proposed system offers a flexible interface along with a search algorithm that supports keyword queries. Our approach is designed for specific classes of databases. We assume that a database to be processed in this approach should represent a particular object, where an onlyspecific type of information within certain tables, fields, and records will be eligible for keyword searchand considered the 'public' ones.

The rest of this paper is structured as follows: Related works are reported in section 2. The proposed approach is described in section 3. Finally, section 4 concludes the work.

2. Related work

Different approaches and techniques for enabling keyword search over relational database have been suggested. For instance, *DISCOVER, DBXplorer*, *BANKS*, and *BLINKS* are the most well known online query systems that will allow keyword-based search on relational databases. Most of these systems usually share a common idea: queries are processed by a graph traversal that searches for connected tuples containing the query keywords.

DISCOVER [10], and DBXplorer [1] are approaches that view the database as a schema graph with tables as nodes, and relationships as edges. These methods construct and evaluate a set of all join trees (called candidate networks in DISCOVER) for a given query, then the potential answers are identified by collecting relevant tuples based on the join trees and schema graph of the database [21].

Other approaches that based on the Steiner trees such as BANKS [11] and BLINKS [9] are usually convert the whole database into a data graph. In such approaches, nodes are denotes to tuples while directed edges are connecting the tuples based on "foreign key → primary key" relation.Methods in such approaches, identify the Steiner trees from the whole graph. A Steiner tree is a connected tree in which every leaf node forms a tuple containing at least one query keyword, and internal nodes correspond to tuples that connect the leaf tuples.

For instance, BANKS identifies the Steiner trees by utilizing a backward search strategy, while BLINKS proposed a bidirectional expansion which can improve the search efficiency comparing to BANKS system.
The core concept of most online keyword query approaches is that modeling the database as a data graph or schema graph. During the searching process, the system traverses the graph to return sub-graphs (Candidate Networks or Steiner trees) as query results. Frequent table joins while processing queries could be extremely costly [20]. The high cost of the online table joins in the online query techniques is considered one of the most drawbacks. These problems could be solved throughout the preprocessing of tables
and tuples in databases [6], [15].
Recently, the off-line query has been discussed deeply, mainly the advantages and the drawbacks of such approaches. The concept of text objects and virtual documents are proposed by some of these approaches, where multi-joining of tables is done offline which improve the query efficiency [19]. Feng et al. improve the text objects by grouping the tuples with the same attribute values to create tuple units data structure [6]. Multiple tuple units could be integrated to achieve a high performance respond to a keyword query.
The above methods only consider the simple table joins and uses SQL-based methods to create virtual documents or tuple units. So, they are not suitable for large-scale databases with complex schemas.
In the meanwhile, a major challenge for keyword search over graph data is query efficiency, which is mainly related to the ranking strategy. For instance, some ranking strategies score an answer by the sum of edge weights. Other approaches, like BANKS, combines two types of information in a tuple tree to compute a score for ranking: a weight of each tuple, and a weight of each edge in the tuple tree that measures how related the two tuples are [20].
Based on the offline query methods mentioned above, we define a practical data graph structure that suites small-scale databases. Such an approach proposes a new ranking method into the computation of ranking scores in a straightforward manner.

3. Proposed approach

Our proposed approach could be described as follows. Given a relational database or any structured data set and its preprocessed collective relation along with the associated inverted list. For any keyword query, we view our approach as retrieving the top k-tuples (within the collective relation) that satisfy the user query. The main idea of this paper could be stated as follows:

1) *Offline Data Preprocessing*: The offline data preprocessing includes mainly two tasks: Creating a collective relation and Construction of appropriate indexes.
- Create the collective relation that represents the portion of data within the database that are eligible for keyword search with a specific ID number for each record in the relation.
- Index all table fields in the collectiverelations<*word,[row- ID LIST]).*
2) *Searching Process*: After submitting queries to the query processor, all indices will be searched based on the query keywords. The searching process includes the following:
- For every keyword match found in an index, create a vertex in a graph data structure.
- For every possible combination of two distinct vertices, create edges between them if the ID numbers associated with each vertex has at least one ID-value (row) in common.
- Apply the proposed search algorithm to find all possible combination for all keywords (vertices) present in the graph by intersecting all ID-numbers associated with the edges adjacent to each vertex.
- Sort the result (of the previous step) according to the number of keywords.
- Select the top k-results according to a predefined k.
3) *Query Results*: ID numbers that resulted from the intersection of edges are used to create SQL queries, and show the top k-results.

In the following sections, each of these steps will be presented in more detail and will be demonstrated with a simple real example database. The small-scale database example is chosen for demonstration purposes and for being proper to the illustration of the main idea of the proposed keyword search algorithm and its functionality.

3.1. Collective relation

In the proposed approach, we presume that the relational database to be processed is concerning a particular type of object, where an only specific type of information within certain tables, fields, and records will be eligible for keyword-based search and considers the 'public' ones. This is the type of database and the class of data that we might suppose to locate on the web and is the sort of database that we are taking into consideration the keyword search problem for.
Data portions selection that needed to provide our proposed process is not novel. Several applications involve squeezing data from multiple relations and tables into a single table to meet the processing necessities, Data Mining processes for example. Data mining techniques depend on looking for patterns of data in a single table. While data in actual databases typically located in many joined tables, a great effort has to be spent in data preparation to compress and structure as much as possible of data into a single table [5]. Such process usually considered as part of pre-processing or cleaning steps. Different works have been proposed, seeking to facilitate data analysis of enormous data sets and to assist in exploring interesting data subsets to process. Attribute selection problem is part of these works [3].
Other approaches that presented to deal with the complex data manipulations of relational databases is data management, which is a straight manipulation of raw data files. Such as extracting partial data directly from a file, carrying out any necessary conversion and writing the outcome to another data file [17].
In our proposed approach, the collective relation supports dealing and representing the structural information simply and easily, since no table joins will be involved, and the relationships between tables will not be taken into account. Consequently, the indexing process will be much easier.
The proposed approach visualize each database as about something in specific (Cars, books,,,etc.) where the collective relation is the probably unnormalized representation about that object. All other tables and attributes are not part of it.
Creating a collective relation concept in our proposed approach is relatively similar to work on universal relations [19], where a database is viewed as a single universal relation for querying manipulations, therefore hiding the complexity of schema normalization. The main

difference between the collective relation in our approach and the universal relation concept is that in our approach the collective relation contains only the data that is selected to be publicly available and searchable, not all data stored in a database as in universal relations. Therefore, creating the collective relation that combines all the required information wanted to be public and searchable is considered the first step in our approach. Such a task should be achievable and should be the responsibility of the database administrator, who has to select the portions of the database that should be included within the collective relation. The administrator should know the searching aims, hence, choosing the necessary data elements should be an easy task.
So, the task starts by generating a collective relation that combines all the data elements within the database that required to be searched. Such a task is related to perform some operations like (outer) join of all publicly accessible attributes that could be possibly renamed. Only tables, records, and fields that may be searched should be chosen while creating this relation. Careful chosen of such data items is a must, such that each of these items is semantically expressive for applying the keyword-based

Table 1: A portion of the collective relation for the example database.

ID	Car brand	Type	Specification	Color	Model	Country	Transmission
1	Mercedes	190e	sedan	Black	1997	Germany	Auto
2	Mercedes	190e	sedan	White	1996	Germany	Manual
3	Honda	Civic	sedan	White	1998	Japan	Manual
4	Toyota	Hilux	pick_up	Black	2001	Japan	Manual
5	BMW	735i	sedan	Red	1993	Germany	Auto
6	Nissan	Patrol	4WD	Blue	1998	Japan	Manual
7	Mazda	323	sedan	White	1993	Japan	Auto
8	Ford	Mustang	sport	Dark Blue	1998	USA	Manual
9	Chevrolet	Caprise	sedan	Black	1992	USA	Auto
10	Oldsmobile	Cultlass	sedan	Blue	1991	USA	Auto
11	Mitsubishi	Pajero	4WD	White	1994	Japan	Auto
12	Audi	180e	sport	Black	1997	Germany	Manual

queries. Many data items including tables, fields and even records will not be included in this relation, such items do not have important data during the search process over the data, especially while applying the keyword query.
Each tuple in the collective relation should be uniquely identified by assigning a unique tuple-ID for it so that we can deal with the data per tuple granularity using a single id. Our assumption here is that the database administrator could get the information above. More details about such techniques can be found in [8]. Our proposed approach will limit its study to the restricted databases classes that will be likely searchable using keyword queries on the web as outlined above. Table 1 shows a portion of the collective relation that created for the example database including fields that eligible for keyword-based search.

3.2. Indexing

The main concern of most information retrieval systems is to provide useful information that the user needs. Traditional information retrieval systems usually use different forms of database representatives that should be accessible to the search system. Such representatives describe the contents of the cuurent database and considered an important requirement to identify the data items within the database to support keyword search.
The most popular database representatives used in keyword search systems and information retrieval systems is the inverted file. Classical inverted file could be defiedas a data structure that connects each word in the dataset to a list of IDs of documents -in the world of text documents- in which the word found, in order to increase the efficiency of the retrieval process. Traditional indexing technique is to create an inverted index. The inverted index usually consists of all terms occurring in the document collection in what is called a dictionary, where each term in this dictionary points to a set of so-called inverted lists, known as posting lists. A fragment of a traditional inverted index is depicted in Fig. 1, where the numbers represent the document id-numbers in which the terms occur [13], [16].

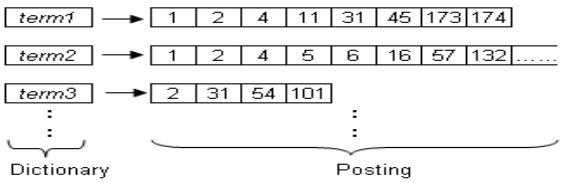

Fig. 1: A fragment of an inverted index.

To serve our proposed approach purposes, we adjust the structure of these inverted files to support keyword search in relational databases environment, whereas traditional information retrieval systems depend on the granularity of documents. These modifications aim to locate the useful information within a relational database or database granularity, specifically at row granularity. So, if a term-n, for instance, belongs to a document with id-m in the traditional information retrieval environment then, according to the scheme of our proposed approach key-n belongs to a tuple with id-m. Fig. 2 shows a portion of an inverted index that extracted from the collective relation of the example database.

{'tuples': [11, 90, 91, 198, 199, 200], 'Key': 'pajero'}
{'tuples': [11, 38, 57, 58, 80, 120, 126, 128, 189, 191, 211], 'Key': '1994'}
{'tuples': [12, 65, 84, 88, 207, 212], 'Key': 'audi'}
{'tuples': [12], 'Key': '180e'}
{'tuples': [13, 13, 210, 211, 213], 'Key': 'volvo'}
{'tuples': [13, 99, 128, 210, 211, 213], 'Key': 'sweden'}
{'tuples': [14, 117, 118], 'Key': '500sl'}

Fig. 2: A portion of an inverted index of the example database.

3.3. Searching indices

Given a keyword query Q that consists of keywords $k_1,\ldots,k_n$, the system searches every index for every kin Q. For instance given the following keyword query "bmw red auto 2009", searching procedure will search the indices for the query keywords and will result in lists with the following contents that depicted in Table 2.

Table 2: Query keywords occurrences from the indices for the given query "bmw red auto 2009"

a)	bmw - [5, 24, 32, 37, 60, 63, 67, 69, 72, 95, 100, 109, 110, ……. 125, 126, 224, 225]
b)	red - [5, 28, 30, 31, 36, 41, 49, 67, 72, 85, 95, 103, 111, …….64, 171, 174, 177, 202]
c)	auto - [1, 5, 7, 9, 10, 11, 13, 15, 16, 18, 19, 20, 21, 26, ….. 213, 217, 218, 221, 222, 225]
d)	2009 - [4, 7, 9, 14, 15, 19, 21, 22, 23, 27, 38, 39, ..……. 200, 202, 205, 213, 215, 220]

3.4. Creating a graph data structure

Ideally, users would like to find the records or the tuples that contain all query keywords are or at least that contain k of all the keywords. So, to retrieve relating data of this kind, we have to work initially with the inter-keyword relationships. Among many alternatives, I choose to use an undirected weighted graph data structure. Such data structure provides an efficient way to handle the complexity of the interrelated data.

If every keyword (after applying a pre-processing step that will be explained later) in a keyword query that occurs in the index with at least one id is declared to be a vertex in the proposed graph data structure. Then, for each pair of vertices, create an edge between them, if they have at least one id in common. Now, the list of the id-numbers that the two vertices have in common will be assigned to the edge between these two vertices as the edge weight. Such a list will have at least one id-number.

By doing so, we construct a weighted graph structure, where all of the keywords query that exists in the indices are represented as vertices in the graph. In the meanwhile, the id-numbers that related to a keyword in the index are related to the vertex of that keyword. Edges will be created between any two vertices if they have at least one common id-number and the edges weight willbe associated with those common id-numbers.

For instance, the example query "bmw red auto 2009" will be processed in the proposed graph data structure as shown in Fig. 2.

The existences of edges between any two vertices are based on the id-numbers part of these two vertices, which means they should have common id-numbers and should be intersected. Intersection outcome will be assigned to those edges. Such assignment is reasonable since edges in this case represent the relationships between the data items according to the id-numbers they have in common.

Suppose we have a set of four vertices {v1, v2, v3, v4}. In order to determine all the pairs of vertices that may be connected by an edge in the graph data structure, we try to intersect the id-numbers associated with vertices by all possible combinations. The following distinct combinations are possible: {v1,v2}{v2,v3}{v3,v4}{v1,v3}{v2,v4}{v1,v4}.

These combinations of vertices should be all tested and intersect because they are potentially related. Table 3 shows the intersection sequence that should be completed for the given the keyword query "bmw red auto 2009".

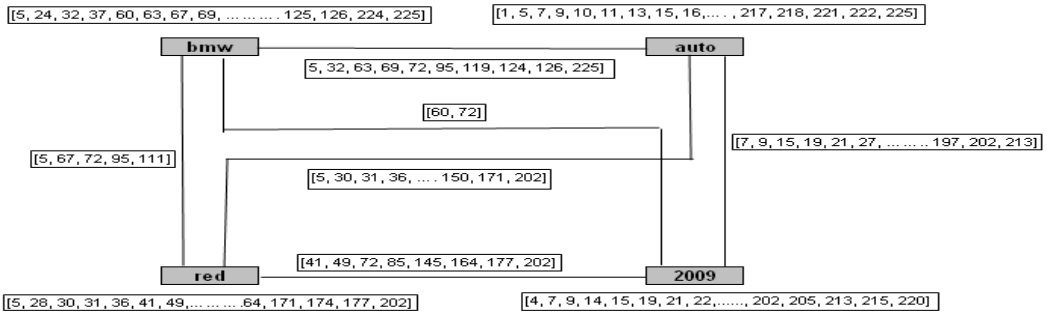

Fig. 2: Undirected weighted graph structure of the example query "bmw red auto 2009"

Table 3: Intersecting vertices for the query "bmw red auto 2009"

[{'tuples': [5, 24, 32, 37, 60, 63, 67, 69, 72, 95, …….124, 125, 126, 224, 225], 'Key': 'bmw'}, {'tuples': [5, 28, 30, 31, 36, 41, 49, 67, 72, 85,…… 164, 171, 174, 177, 202], 'Key': 'red'},
[{'tuples': [5, 24, 32, 37, 60, 63, 67, 69, 72, 95, …….124, 125, 126, 224, 225], 'Key': 'bmw'}, {'tuples': [1, 5, 7, 9, 10, 11, 13, 15, 16, 18, 19,…… 217, 218, 221, 222, 225], 'Key': 'auto'},
[{'tuples': [5, 24, 32, 37, 60, 63, 67, 69, 72, 95, …….124, 125, 126, 224, 225], 'Key': 'bmw'}, {'tuples': [4, 7, 9, 14, 15, 19, 21, 22, 23, 27,…… , 202, 205, 213, 215, 220], 'Key': '2009'}.
{'tuples': [5, 28, 30, 31, 36, 41, 49, 67, 72, 85,…… 164, 171, 174, 177, 202], 'Key': 'red'}, {'tuples': [1, 5, 7, 9, 10, 11, 13, 15, 16, 18, 19,…… 217, 218, 221, 222, 225], 'Key': 'auto'},
{'tuples': [5, 28, 30, 31, 36, 41, 49, 67, 72, 85,…… 164, 171, 174, 177, 202], 'Key': 'red'}, {'tuples': [4, 7, 9, 14, 15, 19, 21, 22, 23, 27,…… , 202, 205, 213, 215, 220], 'Key': '2009'}.
{'tuples': [1, 5, 7, 9, 10, 11, 13, 15, 16, 18, 19,…… 217, 218, 221, 222, 225], 'Key': 'auto'}, {'tuples': [4, 7, 9, 14, 15, 19, 21, 22, 23, 27,…… , 202, 205, 213, 215, 220], 'Key': '2009'}.

The number of intersections increases fast in relation to the number of keywords in the query. For n keyword in aquery,the number of intersections will be (1+2+3+…(n-1)). Regarding the previous query example "bmw red auto 2009" with four keywords, the number of intersections is six as shown in Table 3. A query of 10 keywords needs 45 number of intersections, while a query of 20 keywords needs 190 number of intersections. This shows the increasein the number of keywords will muchincrease the number of intersections, which also will increase the retrieval time.

3.5. Searching algorithm

The system allows the user to insert any number of keywords. To construct the initial graph structure and starting the search process, pre-processes may needed to accomplish the ultimate goal efficiently. During pre-processing, the system should handle the following:

- Extract each word (from the inserted keywords)
- (May apply stemming and stopword removal algorithm.)
- Check for non-existing words in the inverted file and remove them from the graph.
- Check for isolatedwords (of degree = 0) and remove them from the graph. Such process is optional, since isolated words have no common tuples with any other keywords in the query, and could be ignored. Although, we could keep such words if we are interested in all possible results.

It is important to mention here that, in the fortune case, all keywords in the query are related. In such cases, all keywords have common id/s. So intersect the numbers associated with all edges,and we get the final result with these common id/s.Since this is not the general case, for instance, suppose there are three vertices A,B,C and there is a common id between A and B, and a common id between A and C but there is no id between B and C then empty list will be resulted. So, we should traverse the graph to test all possible combination.

The graph could be traversed in many ways. The most important issues here is that a condition should besatisfiedduringtraversing the graph. Before creating a new edge, the new edge must have a common id –at least one- with the latest edge created before this new edge. Fulfilling such condition enables the system to continue traversing on the new edge. We should test all possibilities to come up with the final result. The best-fit answer for a given keyword query is the one that contains as many visited vertices as possible. The path that resulted from traversing yields the id-numbers that spanning the most keywords.

The proposed searching algorithm Fig. 3 could efficientlyanswer a top-*k* keyword query over the collective relation, which constructed to be the database representative of a small scale relational or structured database. The algorithm receivesas input a set of keywords and its associated index list, which will be represented as the vertices in the graph G. The algorithm should produces as output a set of subgraphs that has a top-*k* keywords. Subgraphs includes lists of id numbers (tuples-id in the collective relation) that includes the keywords, and will be ranked by their total score for the query.

Searching Algorithm

Input: A graph $G = (V,E)$
Output: A maximal subgraph components

1. for every v in G do
2. TS(v) = ∅
3. d(S) = 0
4. Determine The**neighbourhood**of*v*, $N_G(v)$
5. TS(v) = {v}
6. d(S) = 1
7. for each u in $N_G(v)$
8. Acw = w(vu)
9. add u to TS(v)
10. d(S) = d(S) + 1
11. if (Acw , d(S)) Not in S(v)
12. add (Acw , d(S)) to S(v)
13. for each r in ($N_G(v)$ – u)
14. Acw = Acw∩ w(vr)
15. if Acw≠∅
16. add u to TS(v)
17. d(S) = d(S) + 1
18. if (Acw , d(S)) Not in S(v)
19. add (Acw , d(S)) to S(v)
20. G = G – {v}
21. return S(v)

Fig. 3: The searching algorithm.

Such that:

- v = v(key,[tuples-List]) [key is a keyword query]
- w(uv) = v([tuples-List]) ∩ u([tuples-List]) [thecommon tuples of v & u]
- $N_G(v)$ = denotes The neighborhood of *v*
- *TS(v)* denotes a temporary subgraph of *G* that containsa node *v*, along with nodes that have common tuples
- S(v) denotes a set of all subgraphs components of G that contains a node v, where the nodes of each subgraph have at least one common tuple.
- *d(S)* is the number of nodes in the subgraph
- Acw denotes the accumulative of the common tuples while traversing the graph

3.6. Creating SQL queries

As a result of applying the search algorithm, and depending on the existence of the query keywords in the database indices, an answer to a keyword query could or could not be found. If the result list of the id-numbers is not empty (top k-results), then the search system gets all it needs to retrieve all relating data (records) from the relational database or specifically from its representative, the collective relation. Top-k results are intended for retrieving top-k records from the database which are including the highest number of queries keyword within the relation [18], which is the collective relation in our approach. The search application will create an SQL query based on the id-numbers of the result list. For our keyword query example "bmw red auto 2009", the result list that returned by the search system is shown in Fig. 4, and the result of the SQL query is shown in Fig. 5.

TOP K-Tuples ----------

{'No. of keyword': 4, 'Tuples-No': [72]}
{'No. of keyword': 3, 'Tuples-No': [49, 72, 145, 202]}
{'No. of keyword': 3, 'Tuples-No': [5, 72, 95]}

Fig. 4: The result list of the search system for the example database.

Query1							
ID	**Car brand name**	**Car type**	**Car specification**	**Color**	**Model year**	**Country**	**Transmission**
5	BMW	735i	sedan	Red	2007	Germany	Auto
49	Nissan	Bluebird	sedan	Red	2009	Japan	Auto
72	BMW	320i	sedan	Red	2009	Germany	Auto
95	BMW	Z7	4WD	Red	2010	Germany	Auto
145	Chevrolet	Camaro	sport	Red	2009	USA	Auto
202	Nissan	Pathfinder	4WD	Red	2009	Japan	Auto

Fig. 5: The result of the SQL query for the example database.

4. Conclusion

In this paper, a graph-based system that enables users to search relational databases using keyword queries was presented. A query in our proposed technique is simply a list of keywords, while the answer to the query is a list of ranked tuples that includes the top-k keywords. Our proposed approach limits its study to the restricted databases classes that will be likely searchable using keyword queries on the web as outlined in section III, which needs offline data preprocessing. To construct the graph, we introduced a modified function for computing edge weightswhich measure the connections among vertices in the graph. Such function is based on the common id-numbers within the lists that associated with these vertices. We conduct a detailed example to show the feasibility of our approach.

Acknowledgment

The work described in this paper is supported by the Scientific Deanship of Zarqa University.

References

[1] Agrawal S, Chaudhuri S & Das G, "DBXplorer: A System for Keyword-Based Search over Relational Databases", in Proceedings of the 18th International Conference on Data Engineering, (2002), pp: 5-16.
[2] Baid A, RaeI, Li J, Doan A & Naughton J, "Toward Scalable Keyword Search over Relational Data", Proceedings of VLDB Endowment, Vol.3, No. 1-2, (2010), pp: 140-149.
[3] D'zeroski S & Raedt L, "Multi-Relational Data Mining: a Workshop Report", in Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-02), Vol. 4, No. 2, (2002), pp: 122-124.
[4] Date C, an Introduction to Database Systems, Wesley, Publishing Company, 1994.
[5] Dzeroskiand S & Lavrač N, Relational Data Mining, Springer-Verlag Berlin Heidelberg, 2001.
[6] Feng J, Li G & Wang J, "Finding top-k answers in keyword search over relational databases using tuple units", IEEE Trans Knowl Data Eng. Vol. 23, No.12, (2011), pp:1781-1794.
[7] Goke A & Davies J, Information Retrieval: Searching in the 21st Century, John Wiley and Sons, (2009).
[8] Hassan M, "Practical Free-Form Search over Relational Databases", UACEE International Journal of Computer Science and its Applications, Vol. 2, No. 3, (2012), pp: 169-173.
[9] He H, Wang H, Yang J & Yu P, "BLINKS: ranked keyword searches on graphs", in Proceedings of SIGMOD 2007: ACM SIGMOD International Conference on Management of Data, (2007), pp: 305-316.
[10] Hristidis V & Papakonstantinou Y, "Discover: keyword search in relational databases", in Proceedings of the 28th international conference on Very Large Data Bases, (2002), pp: 670-681.
[11] Hulgeri A, Nakhe C, Chakrabarti S & Sudarshan S, "Keyword searching and browsing in databases using BANKS", in Proceedings of the 18th International Conference on Data Engineering,(2002), pp. 431-440.
[12] Jia X, Hsu W & Lee M, "Target-Oriented Keyword Search over Temporal Databases", in Proceedings of the 27th International Conference on Database and Expert Systems Applications, (2016), pp: 3-19.
[13] Kanchan D & Paikrao R, "Survey paper on Generalized Inverted Index for Keyword Search", International Journal of Engineering Research and Development, Vol. 10, No. 4, (2014) pp: 69-73.
[14] Kargar M, An A, Cercone N, Godfrey P, Szlichta J & Yu X, "Meaningful Keyword Search in Relational Databases with Large and Complex Schema", Proceedings of the IEEE 31st International Conference on Data Engineering (ICDE), (2015), pp: 411-422.
[15] Li G, Feng J. & Wang J, "Structure-aware indexing for keyword search in databases", in Proceedings of ACM 18th International Conference on Information and Knowledge Management, (2009), pp. 1453-1456.
[16] Manning C, Raghavan P & Schütze H, "Introduction to information retrieval", Cambridge University press, 2008.
[17] Mitha F, Herodotou H, Borisov N, Jiang C, Yoder J & Owzar K, "SNPpy-Database Management for SNP Data from Genome-Wide Association Studies", PLOS ONE, 2011.
[18] Neethu V & Rejimol R, "A Survey of Techniques For Answering Top-k Queries", International Journal of Advances in Computer Science and Technology, Vol. 2, No. 2, (2013), pp: 7-13.
[19] Su Q & Widom J, "Indexing relational database content offline for efficient keyword-based search", in Proceedings of 9th International Database Engineering and Application Symposium, 2005, pp. 297-306.
[20] Wang H & Aggarwal C, "A Survey of Algorithms for Keyword Search on Graph Data. Managing and Mining Graph Data. Advances in Database Systems", Vol. 40. (2010), pp: 249-273.
[21] Wang Y, Wang N & Zhou L, "Keyword Search in Large-Scale Databases with Topic Cluster Units", Technical Gazette, Vol. 25, No. 3, pp: 748-758, June 2018.
[22] Zeng J, Huang J & Yang S, "Top-k Keyword Search over Graphs Based On Backward Search", in Proceedings of the 4th Annual International Conference on Information Technology and Applications (ITA), (2017), pp: 1-6.

Author profile

Mohammad Hassan received his BS degree from Yarmouk University in Jordan in 1987, the MS degree from University of Jordan, in 1996, and the Ph.D. degree in Computer Information Systems from Bradford University, UK in 2003. He is working as an associate professor in the Department of Computer Science at Zarqa University in Jordan. His research interest includes information retrieval and database systems.