Keyword Aggregate Query Based on Query Template

Bin Zhu, Fang Yuan*, Yu Wang
College of Mathematics & Computer Science
Hebei University
Baoding, China
E-mail: bbylshuai@126.com, yuanfang@hbu.edu.cn(corresponding author)

***Abstract*—For keywords query, we propose a keyword aggregate query method based on query template. During the keywords processing, symbol table is used to locate the position of the keywords in the database to get a series of query items. In the query template generating stage, we design a heuristic query template generation algorithm. We use the improved scoring rules to rate the query templates, and select the optimal query template. The experimental results have shown that the aggregate query method is effective.**

Keywords- aggregate query; query template; query item; database

I. INTRODUCTION

Along with the extensive application of database technology, a large amount of information is stored in the database. In order to retrieve the required information from database, the users must not only master complex grammar specification, but also know the mode information of database. However, it is difficult for the general users who have no background of database knowledge. The advantage of information retrieval technology is that it is simple and convenient, the information can be obtained just by keywords. Information retrieval techniques are applied to the query in the database, we can access database through keyword query, just as using a search engine to retrieve the information, which make it intuitive and easy-to-use for users to access databases.

Keyword query which is implemented in database can simplify the query process, reduce the difficulty and complexity, and furthermore, it is based on the entire contents of the database, without the restrictions of relationships and properties the generated result is not just confined to a single attribute, a tuple or a single table, we need to connect the structure that contains different keywords information in the user query, maybe it is a tuple connected by multiple tables, expanding the coverage of the query. In this background, the research about database queries based on the keywords has become one of the hot issues, which attracted scholars' attention in different areas, such as information retrieval, Internet and database.

In this paper, according to the keywords query in the database, we proposed a of query template-based aggregation query method. When processing the user-specified keywords, the symbol table [1] is used to locate the position of the keywords in a relational database, forming a query term. When generating query templates, we design a heuristic query template generation algorithm. In the query result generating, we exploit the improved scoring rules to rate the generated query templates, and then, select the optimal query template to generate results. Finally, the experimental results show that the aggregate query method is effective.

II. RELATED WORK

In recent years, the most scholars conduct their own research on database query based on keywords. Aiming at relational database, [1] has implemented search engine DBXplorer based on keywords. For a given set of query keywords, DBXplorer return all rows containing all keywords in the database, these rows are from a single table or multiple tables joining. Reference [2] put forward the idea of sorting the search results, then, achieved a framework for keyword query and proposed a heuristic algorithm for the incremental traversal of query results, and implemented the BANKS system that supported relevance ranking and result presentation. Reference [3] made a further study of the architecture and algorithm optimization on the basis of the [1, 2], and put forward the DISCOVER system, it can found all candidate networks by utilizing the database schema without reproducibility. Then, [4] applied the sort strategies in the field of Information Retrieval to the database search, and proposed IR-Style sorting strategies. Reference [5] expanded the database search language, the SEEKER systems based on three kinds of keywords not only can support the search of the property value, but also can search the metadata and digital properties, and improved the correlation score mechanism, then returned the top-k results to the user. Reference [6] proposed adding phrase recognition into the query algorithm, and increasing the weight of important phrase in the user queries, which can makes them be in the forefront of the result, satisfying the user requirements. Reference [7] proposed a new idea called combination query to answer keyword query by returning tuple combinations. Reference [8] aggregated the tuple of the results which obtained by the previous keyword query technology to achieve the purpose of aggregate query.

The above literatures focused on finding a set of tuples that best match keywords, that is, a tuple from a table or connecting tuple from multiple tables. However, up to now, the vast majority of research can only support simple keyword query, without supporting more complex aggregation query. In order to obtain the results of aggregate query, the user still need to learn the SQL statement and the

Published by Atlantis Press, Paris, France.

knowledge of database schema, to construct an appropriate SQL aggregate query statement. Although the keyword search which supports the aggregate query has superior practicability, but the corresponding achievements are less. Reference [7] proposed a new idea called keyword query based on tuple combination, it can be considered as special aggregate query. It was overall tuples of the entire table, rather than the form of a single tuple, but it was not a real sense of the aggregate query. Reference [8] aggregated the tuple of the results which obtained by the previous keyword query technology, but it can not specify the aggregate query by the user.

Although using the single tuple to answer the query is very useful, sometimes we need to use aggregate query. Therefore, in order to better satisfy the user requirements, we should be able to support aggregate keyword query. But sometimes, returning a tuple of a table, or connecting tuples from multiple tables, or the existing aggregation keywords query technology, can't solve this problem, the users might be interested in aggregate keyword query of their input.

In this paper, aiming at the existence of the above problems, we proposed a new method called keyword aggregate query based on query template. During the processing of keywords, we use the symbol table [1] to locate keywords, and form the query item which can express keywords. When generating query templates, we design a heuristic query template generation algorithm, for each query term, form a query template; we use the improved scoring rules to rate the query template which is generated by query item, and select the optimal query template to generate results.

III. KEYWORD AGGREGATE QUERY

In this paper, keyword aggregate query has been divided into three stages for processing. The first stage is keywords preprocessing, the keywords is located in the database during this stage, generating the query items. The second stage is generating the templates, by which we can get the query templates. The third stage is rating the results for the query template.

A. Keywords Processing

Keywords Aggregate query is consists of a set of keywords like k_1 ,k_2 ,⋯, k_i ,⋯ ,k_n , denote it by Q (k_1 k_2 ⋯ k_i ⋯ k_n), and existing one keyword k_i as aggregate keyword(count, min, max, etc, called aggregate function name).

It is necessary to perform the corresponding preprocessing for the user inputting aggregate keywords, recognizing different kinds of keywords, and representing them effectively. Learning from the classified processing for the query keywords in [5], we define the query item, and represent the query keywords which user input by query item.

Definition 1: (Query Item QI) The specified relational database D include a series of tables such as T_1 ,T_2 ,… ,T_i ,… , T_n,, and there are a set of columns called $C(t_i)$ for T_i. A query item is a triples, represented by QI= (A, a, F). The definitions for these words "A", "a", "F" are as follows:

① $A = \{(a_i^j, \text{value})\}$, a_i^j is the j-th column of the i-th table, or called "(table name).(column name)", and "value" is the value of a_i^j.

② a ∈A, means aggregating on "a".

③ F is a aggregation function.

It is inevitable that there might be some meaningless query items, learning from the idea of [7], we propose a new definition called Common Query Item.

Definition 2: (Common Query Item) If there is a query item among following situations, we will call this query item as common query item.

①There are two columns in A called a^i , a^j , which refer to the same attribute.

②According to the function dependency relationship, if there is a relationship like b ∈ A, b ≠ a, then b → a.

③ There is no aggregate function F in query item.

If the query item doesn't satisfy any one of the above situations, then we call this query item as non-common query item, or important query item.

It is necessary to analyze the query keywords committed by user. And then recognize all kinds of keywords and store them. It needs to recognize the effective aggregate keywords meanwhile analyzing the query keyword. The keywords which have been recognized should be matched with the mode of database (including table name, column name) respectively.

In this paper, we adopt the approximate string matching algorithm [9] to define the matching degree between keywords and elements of query item. Every keyword has a matching degree obtained by the approximate string matching algorithm. If the matching degree between the mode element and the keywords is higher than the given threshold, then it can be regarded as a potential match. After using the matching algorithm, each keyword can generate a series of potential matches and form a matching table. Then it will produce a series of query items through cross product of each keyword matching table. It is inevitable that there might be some meaningless query items, so it is necessary to identify the meaningless query item by using the common query definition and functional dependences. Common query item should be eliminated before forming SQL query.

For example, consider that the keyword of aggregate query is "Bill num paper". It is trying to calculate how many total papers have been published for Bill. This keyword can generate many query items and one of probable query item is as follows:

({Author.name = [Bill], Paper.paperid}, Paper.paperid, count). Certainly, there are some other probable non-common query items.

B. Query Template

Query Structure is necessary for SQL query, because it specifies the entities involved in the query and the connection type between them. From the perspective of the SQL grammar, query structure is included in the relational entities used in FROM clause and WHERE clause. The existing research on data query has put forward a lot of query

structure simulation methods. The relatively familiar query structures used in the relational database query methods are join tree of tuples [1], joining networks of tuples [3], Candidate Network [4, 10] etc.

However, aggregation has not been considered in the existing query structure. From the perspective of SQL, the aggregation query aggregates on the relational attributes, but the current query structure couldn't express this point accurately. Thereby, it is necessary to redefine a query structure which is suitable for aggregate query.

Definition 3: (Schema Graph) [1,3,4] Assume that the database called D has n relations denoted by R_1, R_2, ⋯, R_n, and each relation R_i has m_i attributes represented by $a_1^i, a_2^i \ldots, a_{m_i}^i$. The database schema graph (G) is a directed graph, which reflects the primary-foreign key relationships in the database schema. The relation R_i has corresponding node in G. If there are some primary-foreign key relationships between attributes $\{a_{b_1}^i, a_{b_2}^i \ldots, a_{b_k}^i\}$ and attributes $\{a_{b_1}^j, a_{b_2}^j \ldots, a_{b_k}^j\}$, then there will be a corresponding edge between node R_i and R_j in schema graph G, we can describe it as $R_i \rightarrow R_j$.

The relations in the database are represented as nodes in schema graph. The relationship between primary key and foreign key are represented by the directed edge, which starts from the primary key and ends with the foreign key. DBXplorer [1], DISCOVER [3], IR-Style [4] all adopt this method. Fig. 1 is the DBLP [11] database schema graph.

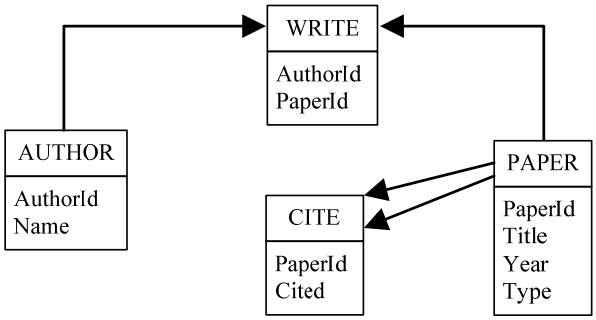

Figure 1. The DBLP database schema graph.

Combine with the idea of query structure in [1,3,4,10], we propose Query Template to simulate the query structure according to the characteristics of aggregation query and schema graph. Consequently, we create query template according to the database schema graph G and non-common query items.

Definition 4: (Query Template) Query Template is the extension for the connected subgraph G_T (V_T, E_T) of schema graph G, where V_T signifies the node set, E_T denotes edge set. The definition of V_T and E_T are as follows:

Node set V_T: It includes entity nodes (relation tables) or attribute nodes (attributes of the entities).

Edge set E_T: It includes the undirected edges between entity nodes and attributes node or the directed edges among the entities with primary-foreign key relationships.

If the template includes only one entity node, we call it atom template.

A query template can be considered as a join expression used to generate potential query results. Learning from the require for query structure in [3,12], an efficient query template must satisfy the following properties:

Aggregation: There must be a node labeled as aggregate node, meanwhile existing in the query items.

Minimality: Any node can't be removed from the query template, if any node is removed, the remaining nodes couldn't be forming a query template with integrity.

Integrity: The node which expressed by "A" in query item appears at least one time in query template.

Uniqueness: The exactly same query template QT is forbidden.

Controllability: The number of nodes should not exceed the default value MaxQTSize which is the maximum permissible size of query template. (Setting template's maximum is mainly for large and complex database)

For instance, Fig. 2 provides a query template, and according to the corresponding query template, it can search the total number of a certain thesis which published in a certain time. The query template has three entities which include Paper, Author, Write and the used attributes. We can see that the join between entity nodes is based on the primary-foreign key relationships. The edge starts from the primary key node and ends with the foreign key node. However, the join between entity nodes and attribute nodes are denoted by undirected edges.

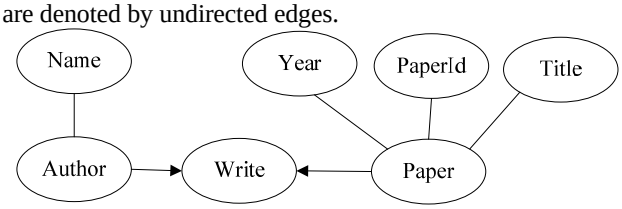

Figure 2. An example of query template.

Learning from the heuristic thoughts, we put forward a heuristic query template generation algorithm (CreateQT algorithm). It analyzes the query items and then provides the corresponding query template.

The basic idea of the CreateQT algorithm is: The initial state of the algorithm is a temporary solution "temp" which only includes an aggregate node. Then search a node in query items which has the shortest distance between "temp" in the schema graph. Next, add the searched node and the path between the node and "temp" to the temporary solution. The algorithm calculates the distance between the nodes in query items and "temp" repeatedly, and continue to add the next shortest node and the corresponding path to the temporary solution. When all the nodes of query item are covered, the algorithm will be terminated. However, if the algorithm cannot be able to be backtracking, and does not find the next solution, the algorithm will be terminated and reports an error.

Algorithm 1: Query Template Generation Algorithm (CreateQT Algorithm)

Input: aggregate node, nodes in query item, current query template (temporary solution, Initialized as a node which only includes a aggregate node), schema graph
Output: current query template
Begin
1. if some other nodes in query items are NULL
2. return current query template
/*Initialization*/
3. bool Find = FALSE;
4. temp = current query template (only including aggregate node)
5. while(TRUE)
/* repeatedly positing the nodes which has the shortest distance from the current query template */
6. for each node N in the current template{
7. for each edge E in the current schema graph{
8. if the edge E is included with N, and is the shortest edge. And that, the other end node of E is not in current template {
9. add edge E and the other node E' into current template;
10. Find=TRUE;}}
11. if schema graph does not include the node, add a undirected edge connecting the node and related node}
12. if Find = FALSE return NULL;
/*It does not find the next add solution */
13. search the nodes of query item in current template matching nodes
14. if the matching nodes is null then continue;
15. Add = FALSE;
16. for each node in the matching nodes{
17. if it fulfill several conditions of the query template{
18. Add=TRUE;
19. add the nodes and the related path into the current template;
20. remove the nodes and the path in the query item;}}
21. if (Add = TRUE){
22. CreateQT(aggregate node, nodes in query item, current query template, schema graph) ;}
23. else continue;
24. end while
End

The query item which is generated in the preprocessing stage of keywords can form corresponding query template. A query template will be uniquely corresponds to a structured query.

C. Generate the Query Results

The work before delivering final results to the users is to sort the query template, because there are many possible query templates meeting the keywords matching conditions. However, users are usually only interested in the most relevant results.

The score of one query template is depending on the nodes weight and edge weight. At first, the scoring function assigns one score to each query template to measure the matching degree between query results and query keywords. The result with the highest score will be returned firstly. The ranking function is based on the scoring function. It ranges the query template according to the scores.

For the node weight, we learn the method which is proposed in SEEKER [5] and improve the method to calculate the similarity between query keywords and query template. In the method proposed by SEEKER, the nodes in every query template should be matched with every keyword. In this paper, each node in query template is corresponding with each query keyword. The number of the keywords in query results is not identical. Obviously, the more contained keywords, the higher the score. The improved scoring formula is as follows:

$$Score(QT,Q)=\frac{1}{sizeof\,(QT)}\left(\frac{m'}{m}\right)^{a}\sum_{i=1}^{sizeof\,(QT)}Score(T_i,k_i)\cdot \qquad (1)$$

Where, QT is a query template which is consists of query results. "sizeof(QT)" is the number of nodes which QT contains. T_i is the node in QT; Q is one of the keywords query, m is the number of keywords in Q, and $k_i \in Q$; m' is the number of the nodes in QT; "a" is constant to ensure that the score of query results which contained more keywords is higher than the ones with less keywords; $Score(T_i,k_i)$ is a matching degree for a given keyword with the corresponding node. The value of $Score(T_i,k_i)$ is standardized in the [0, 1] range, where 0 means not match, 1 means exact match.

For edge weight, in the query template, the adjacent relationship of a single node with the surrounding other nodes reflects that this node's impact on the surrounding nodes and the importance of the its content. The adjacent relation is embodied by the in-degree of nodes, if the in-degree of one node is greater, it indicates that the primary key of this node is the foreign keys of more other nodes. Consequently, the more the node should be included in the query template formed by query keywords, that is, the adjacent edges with this node should be given priority to be accessed during query time.

In order to reflect the importance of node in-degree in edge weight calculation, the following formula [13] is used to calculate the edge weight.

$$w_e(u,v)=\begin{cases}\ln(\lambda_1 w_1+\lambda_2 w_2), & \lambda_1 w_1+\lambda_2 w_2\neq 0\\ 2*\min\text{weight(k,u)}, & \lambda_1 w_1+\lambda_2 w_2=0\end{cases}. \qquad (2)$$

Where, $\lambda=\ln(1+N_{in}(v))$ reflect the impact of the node v's in-degree on edge weight. $N_{in}(v)$ is the in-degree of the node v. w_1, w_2 are the similarity between the nodes connected by the edge e. $\min\text{weight(k,u)}$ is the minimum similarity of all nodes in the query templates. From (2) we can see that the greater $\lambda_1 w_1+\lambda_2 w_2$, the greater the weight of edge e, and then the edge has a higher access priority.

We use (1) and (2) to calculate the score of query template, and sort them, the query template with the maximum weight score will be selected as the optimal query template corresponding to the query item.

We use the CreateQT Algorithm to find query templates, rate and sort them. Next, convert the query template in which

the users are interested into the corresponding structured query statement. Finally, submit these query statement to DBMS to run and return the query results.

The implementation process of the query results generation is described in Algorithm2.

Algorithm 2: Results Generation Algorithm (TranslateQT algorithm)
Input: query item, query template
Output: Structured Query Statement (SQL)
Begin
1. The elements in query item→SELECT clause;
2. The entity nodes in query template→FROM clause;
3. The edge between the entity nodes in query template→WHERE clause;
4. The element value in query item→WHERE clause;
5. Element in query template(except aggregate nodes) →GROUP BY clause;
6. return SQL statement.

End

IV. EXPERIMENTS

A. Experiment Setup

The experiments are running on a PC with an Intel Pentium 4, 3.06 GHz CPU and 2G RAM with 160GB disk, running a Windows XP operating system, and all the algorithms use VC + + language to connect database SQL Server 2000.

In order to evaluate the performance of the aggregate query method, the experiment is performed on DBLP [11] dataset. The DBLP dataset source is XML file which is provided by DBLP website. We use these XML data to construct the DBLP schema graph as shown in Figure 1, load the data mappings into relational database, and decompose them into four relationships: Author (Recording the authors information), Paper (Recording the paper information, such as published in which year and in which International Conference), Write (Recording the information author writing paper). Cite (Recording the referenced information about one paper citing another paper). The dataset is stored on SQL Servers.

B. Query Effect

In this paper, we use Precision and Recall to measure query effect. In the experiment, we use keyword query to finish the same task as SQL query. According to the query feedback record, we get the query result set matching with the user query and use it as a reference set. And we calculate the keyword query precision and recall based on the query result of reference set.

At first, we construct five groups of keywords aggregate query testing set Q_i (i=3, 4, 5, 6, 7, represented as Q_3 - Q_7) ,each group has 12 queries, "i" is the number of query words contained in this group of query, and the query is about DBLP dataset' s aggregate query.

TABLE I. EXPERIMENTAL RESULTS OF THIS PAPER

Keywords Query	Precision	Recall
Q_3	81.2%	85.6%
Q_4	78.3%	82.9%
Q_5	81.5%	80.6%
Q_6	74.7%	77.7%
Q_7	72.5%	73.6%
Average	77.6%	80.8%

Table Ⅰ describes five groups of testing set query results about keywords aggregate query, and provides the Precision and Recall. As seen from Table Ⅰ, for each Q_i, with the increasing of the keywords number, the number of query template associated with query in database is usually also increased the query returning results will become a smaller fraction of the total matching results. So the Recall will be reduced with the increase of the query keywords number.

Comparison between method in [8] and our proposed method is as shown in Table Ⅱ.

TABLE II. COMPARISON OF THE METHODS BETWEEN [8] AND THIS PAPER

	Average Precision	Average Recall
Experimental result in [8]	61.8%	79.8%
Experimental result in the paper	75.3%	80.5%

As seen from Table Ⅱ, the Recall of the method used in [8] and in our paper is almost the same, but the Precision of our proposed method is higher than the one in [8], this proves the effectiveness of our proposed method. In [8], the author aggregate the result tuples obtained by previous keywords query technique to achieve the purpose of aggregation query. However, it cannot guarantee that the query results may completely contain all the tuples that meet the requirements, which will affect the accuracy of the results and reduce the precision. In this paper, we propose the concept of query template, and use query template to improve the precision rate while guaranteeing the recall rate.

Finally, combine the above experimental results, it shows that the method has good feasibility, and has a good query performance.

V. CONCLUSIONS

In this paper, in order to solve the aggregate query problem on database, we propose a new method called aggregate query over databases based on query template, it forms query item through the processing of keywords and generates the most probable template of aggregate keywords as the query result. We use the improved calculation method of edge weight and node weight to enhance the query effect. We verify the effectiveness of the proposed aggregate query method through a series of experiments on real datasets.

This method is a single aggregation query, and further research will focus on complex aggregate query.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China under Grant No. 61170039 and Key Lab. in Machine Learning and Computational Intelligence of Hebei Province.

REFERENCES

[1] S. Agrawal, S. Chaudhuri, and G. Das, "DBXpIorer: A system for keyword-based search over relational databases," Proc. of the 18th Int'l Conf J. eds. Data Engineering (ICDE 2002), IEEE Computer Society Press, 2002, pp. 5-16.

[2] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, "Keyword searching and browsing in databases using BANKS," Proc. Of the 18th Int'l Conf. on Data Engineering (ICDE 2002), IEEE Computer Society Press, 2002, pp. 431-440.

[3] V. Hristidis, and Y. Papakonstantinou, "DISCOVER: Keyword search in relation databases," Proc.of the 28th Int'l Conf. on Very Large Data Bases (VLDB 2002), Morgan Kaufmann Publishers, 2002, pp. 670-681.

[4] V. Hristidis, L. Gravano, and Y. Papakonstantinou, "Efficient IR-style keyword search over relation databases," Proc. of the 29th Int'1 Conf. on Very Large Data Bases, Morgan Kaufrnann Publishers, 2003, pp850-861.

[5] J. J. Wen, and S. Wang, "SEEKER: Keyword-based information retrieval over relation databases," Journal of Software, 2005, pp. 1270-1281.

[6] P. Li, Q. Zhu, A. J. Ren, W. Hu, and X. Y. Du, "Novel algorithms of keyword search over relational databases and phrase recognition," Computer Science, 2008, pp. 134-138.

[7] Y. Tao, Z. Y. He, and J. Q. Zhang, "Keyword queries over relational databases based on tuple combination," Journal of Computer Research and Development, 2011, pp. 1890-1898.

[8] B. Utharn, P. Krirngkrai, S. Umaporn, "Answer aggregation for keyword search over relation databases," Proc of the IEEE Conf on Visual Analytics Science and Technology, 2010, pp. 477-482.

[9] H. M. Wang, Algorithm Design and Analysis, Beijing: Tsinghua University Press, 2006, pp. 132-134.

[10] Y. Luo, W. Wang, and X. Lin, "Spark: A keyword search engine on relational databases," The 24th International Conference on Data Engineering, 2008, pp. 1552-1555.

[11] The DBLP Computer Science Bibliograhy [DB/OL], http://dblp.uni-trier.de/.

[12] J. F. Xi, G. H. Liu, J. C. Li, J. J. Tang, and R. L. Qi, "Top-k oriented hierarchical keyword-based information query system architecture over databases," Journal of Yanshan University, 2004, pp. 67-73.

[13] Y. Zhang, F. S. Jin, G. H. Liu, Y. Yuan, and L. Li, "Minimum steiner tree based method to keyword search," Journal of Chinese Computer Systems, 2010, pp. 119-123.