A New Approach to Compute Structural Statistics using Keywords in Databases

Ambily Jose[1], Jobin George[2], Liliya T Jose [3]

[1]*Dept of Computer Science and Engineering,*

BTL Institute of Technology, Bangalore, India.

[2] *Software Engineer,Bangalore, India.*

[3]*Dept of Computer Science and Engineering,*

Vimal Jyothy Engineering College, Kerala, India.

***Abstract*— Query using keywords is one of the user friendly and widely used forms of querying in recent days. The existing systems focused on fetching all or a fixed number of top tuples , which may directly contain the portion of keywords or interconnected to other related tuples via foreign key. These systems will not give any valuable information more than individual interconnected tuple structures.. In this paper we focus on compute structural statistics for keyword by extending the group & aggregate framework. In RDBMS, rows with same values on specific attributes can be grouped together and aggregate functions can be used to aggregate the values of the same group. Tuples are considered as rooted subgraphs, which represents an interconnected tuple structure among tuples . The keywords are separated as dimensional keywords and general keywords. The dimensions of the rooted subgraphs are determined by dimensional keywords and the general keywords are used to compute scores. Based on the score computed for every group, an aggregate function is used to compute the aggregates.**

***Keywords* - Keyword search, relational database , Structural statistics.**

I . INTRODUCTION

Search based on keyword in database systems is one of the commonly used research topic. It helps the users to get results which match the keywords. Most of the available studies are based on finding the related values based on the values on the table and values on other tables which is referred by a foreign key reference[1]-[3]. These systems may give a large set of possible results for the given keywords, and sometimes users may struggle to filter the required information. In this paper, we are going to evaluate how to compute statistics on the interconnected tuple- structures using keywords instead of finding interconnected structures among tuples.

II. STRUCTURAL STATISTICS

Let GS = {R1,R2, . . .} be a relational database schema. Here,Ri in GS is a relation schema with a set of attributes. Keyword search is allowed on any text-attributes. A relation schema may have a primary key and there are foreign key references defined in GS. We use Ri → Rj to denote that there is a foreign key defined on Ri referring to the primary key defined on Rj. A relation on relation schema Ri is an instance of the relation schema (a set of tuples) conforming to the relation schema, denoted r(Ri). A relational database (RDB) is a collection of relations.

We model an RDB over GS as a directed database graph GD(V,E). Here we use two types of labeled nodes tuple nodes Vt and attribute nodes Va, V = Vt U Va for Vt ∩ Va= Φ. A tuple-node in r(R') labeled with its relation name R'and ti represents a tuple in r(R'). An attribute-node is labeled Aj : ak where Aj is a text-attribute name and ak is an attribute value. Consequently, E = Ett U Eta consists of two types of edges.Ett is the set of edges among tuple-nodes in GD, and an edge ti → tj in Ett indicates that there exists a foreign key reference from ti to tj in RDB. Eta is the set of edges from tuple-nodes to attribute-nodes. There exists an edge from a tuple-node ti to an attribute-node Aj : ak in Eta, if the tuple ti has the attribute value ak in the attribute Aj in RDB.Here V (G) and E(G) to denote the set of nodes and the set of edges of G, respectively.

If ki is contained in either the attribute name or the attribute value in the node label then we can say that there exist an attribute-node v contains a keyword ki.We also say a tuple-node u contains a keyword ki if there is a path from tuple node u to an attribute node v contains Ki.

Virtual Tuple. A virtual tuple is a tree representation of the maximum subgraph at a tuple-node tγ in GD and denoted e as Vtuple, or explicitly Vtuple(tγ) if it is rooted at a tuple-node tγ. The maximum subgraph of tγ is the induced subgraph of all nodes that are reachable from tγ in GD. The tree representation of Vtuple(tγ) is, if a node u links to a node w which is predecessor of u, the edge (u,w) will be deleted. Otherwise, if two nodes u and v link to a node w in GD, we create an additional copy w' of w, and let u link to w and v link to w'. All leaf nodes in Vtuple(tγ) must be attribute-nodes A Vtuple(tγ) includes all information a tuple-node tγ can reach .

Dtree, Gtree, and DGtree. Given a set of keywords {k1,k2, . . .}, a Dtree(tγ) is a minimal subtree of Vtuple(tγ) which contains all dimensional-keywords by connecting to the attribute-nodes that contain the given dimensional keyword. A Gtreer tγ) is also a subtree of Vtuple(tγ) by removing all the subtrees rooted at a tuple node that do not have any attribute-node containing general keywords.A Gtree(tγ) matches a query if it contains at least one general-keyword. A Gtree(tγ) may not contain all the general-keywords. Given a set of keywords, there exists one Gtree(tγ), but many distinctive Dtree(tγ), by definition. A DGtree(tγ) consists of two parts, where the first part is Dtree(tγ) and the second part is Gtree(tγ).

In this paper, we study a new structural statistics keyword query Q = (Qd,Qg ,α,β)against an RDB over GS.It consists of two sets of keywords, namely Qd and Qg, for Qd ∩ Qg = Φ , a score function α, and an aggregate function β. We call a keyword in Qd and Qg as a dimensional-keyword and a general-keyword , respectively. The two sets of keywords, Qd and Qg, together specify a set of trees T to be computed. A DGtree Ti in T with root tγ = root(Ti) consists of two subtrees, a Dtree rooted at tγ for Qd, and a Gtree rooted at the same tγ for Qg, denoted as Dtree(Ti) and Gtree(Ti), respectively. The set of trees, T , are grouped into different groups. Let Ti and Tj be two trees in T . Ti and Tj belong to the same group if Dtree(Ti) is isomorphic toDtree(Tj). Here score function α to be any possible algebraic function based on TF-IDF, namely, tf_w(T) and idfw.. Let T ={T1, T2, . . .} be a set of DGtrees in the same group. We consider every Gtree(Ti) for Ti Є T as a virtual document, by merging all attribute names and attribute values in the tree into a multiset. Then, tf_w(Ti) is the number of times the keyword w Є Qg appears in the corresponding virtual document, and idfw is calculated as follows[5]:

$$idf_w = \frac{|T|}{df_w(T)} \quad (1)$$

where |T | is the number of Gtrees in T in the group, and dfw(T) is the number of Gtrees that contain the keyword w Є Qg in the group. The tree level ranking function[4] is such an algebraic function based on tfw(T) and idfw. The α function is to be applied to Gtree(Ti) for Ti Є T to give such Gtree(Ti) a score. Factors that can be involved in an α function can be, for example, to give a high score for a term if it is close to the root. For efficiency consideration we require that factors in an α function for a tree must be computable from the factors of its subtrees. The aggregate function β aggregates the scores computed for DGtrees in the same group. An aggregate function can be any SQL aggregate functions (min, max, sum, avg, and count). The output for the group T is (TA,ω) where TA represents the Dtree for the group, and w=β({α(Gtree(T1)), α(Gtree(T2)), … }).

III. SOLUTION OVERVIEW

Given a structural statistics keyword query Q = (Qd,Qg ,α,β) over an RDB, a naive solution is impractical for the following two main reasons. 1) The number of possible Dtrees and Gtrees can be very large. It is infeasible to compute. It is worth noting that all the existing solutions focus on finding top-k answers if they make use of ranking and allow some (not all) keywords to be contained in the answers. 2) It is costly to group Dtrees into groups even though tree isomorphism checking is polynomial.

In this paper, to compute a structural statistics keyword query, Q, we propose a two-step approach (Algorithm 1). In the first step, we generate a set of label-trees for dimensional keywords (LDs), denoted as L = {LD_1, LD_2, ..}, such that every DGtree to be computed will conform to a unique LD. In the second step, we compute the structural statistics keyword query Q using L. In brief, for every LDi, we compute all DGtrees, denoted as T i = {Ti_1; Ti_2 ; . . .} that conform to LDi, group all the trees in Ti into groups based on Dtree(Ti_j) and compute α (Gtree(Ti_j)) for every Ti_j Є Ti, and then compute β for every group. The main idea behind label-trees is to avoid tree isomorphism checking and enumerating all possible DGtrees.

The algorithms to generate all label-trees and to compute structural statistics keyword query using the label-trees will be discussed in the following sections.

ALGORITHM 1. Structural-Statistics(GS, Q, GD)

1: L =LD-Gen(GS,Q);
2: for every LDi Є L do
3: LD-Eval(Q, LDi,G_D);

First, we define a label-graph, Gs(V,E), for keyword search for a database schema GS. Here, V is a set of nodes such that V = V_R U V_A, where V_R is a set of nodes labeled with relation names, called relation nodes, and V_A is a set of nodes labeled with attribute names for those text-attributes only. E is a set of edges such that E = E_{RR} U E_{RA}. An edge Ri → Rj appears in RR if there is a foreign key reference from Ri to Rj. An edge Ri → Aj appears in E_{RA} if Ri has an attribute Aj. Second, we define a rooted tree for every relation Ri in Gs, denoted as LV(Ri), which is a labeled tree for all virtual tuples rooted at $t_γ$ Є r(R_i). LV(Ri) is a connected tree representation of the maximum subgraph rooted at Ri in Gs.The tree representation is done as follow. In LV(Ri),if a node u links to a node w which is an ancestor of u, the edge (u,w) will be deleted. Otherwise, if two nodes u and v link to a node w in GS, we create an additional copy w' of w, and let u link to w and v link to w'.

Consider a structural statistics keyword query Q =(Qd,Qg, α, β) against the data graph G_D using the label-graph Gs. We further give a specific LV for computing Dtrees. A label-tree for dimensional-keywords LDi(Rj) is a subtree of LV(Rj) rooted at Rj that contains at most |Qdj| attribute-nodes as leaf nodes. The attribute-nodes in LDi(Rj) possibly contain all the dimensional-keywords in Qd.

IV.GENERATE ALL LDs

The solution is as follows , first, we precompute all LVs for G_S because the set of LVs is query independent. The algorithm to compute all LVs is shown in Algorithm 2 [5]. For each node R in GS, we calculate LV(R) using a breadth first search from R in GS until all nodes that can reach from R are added into LV(R). For node u that is visited more than once in the breadth-first search, we create an extra copy in LV(R) if u is not visited from its descendant. Second, in order to efficiently generate all LDs for all possible dimensional-keywords, we construct an inverted index, called the dimensional inverted index (DII), using the names and values of the attributes in the RDB. The inverted index helps to find the attributes in a relation that a dimensional-keyword di matches. In detail, for each possible dimensional-keyword, w, in DII, there is a list of entries to describe the attributes in a relation the keyword w matches. We denote the list for w as list(w). Each entry e Є list(w) has three fields: e = {Type, Rel, Attr). Here, Type can be either Name or Value. When Type = Name, it means w is an attribute name. When Type = Value, it means w is an attribute value. Rel and Attr indicate the relation and the attribute that w matches.

ALGORITHM 2. LV-Gen(GS)
Input: The label-graph GS (VR U VA; ERR U ERA)
Output: The set of all LVs.

```
1: S  ← Φ;
2: for all R Є VR do
3: add R with links to all its text-attributes into LV(R);
4: Q ← Φ;
5: Q:enqueue(R);
6: while Q ≠ Φ do
7:     R1 ←Q.dequeue();
8:     for all R1 → R2 Є E(Gs) do
9             let R'2be a copy of R2;
10:           add an edge in LV(R) from R1 to R'2
11:           add links from R'2 to to all its text-attributes;
12:            Q:enqueue(R'2);
13: S  ←S U {LV(R)};
14: return S
```

We design a new algorithm to generate all LDs using list(di) for di Є Qd, as shown in Algorithm 3. The algorithm[5] generates all LDs for a structural statistics keyword query. First, it collects information if a keyword di matches the relation nodes in each LV (lines 2-6). Given the set of LVs S, for each LV in S, we use LV.list$_i$ to maintain the set of candidate entries in list(di) that may contribute to generating LDs from LV. Second, in a for loop (lines 7-10), for each combination of nodes $e_1, \ldots, e_{|Qd|}$ that contain keywords $d_1, \ldots, d_{|Qd|}$, respectively, in a certain LV Є S, we generate a LD by constructing a minimum tree that contains nodes $e_1 \ldots e_{|Qd|}$ in LV. Given LV and $e_1, \ldots e_{|Qd|}$, the LD is unique and can be computed as follows: for each leaf node of LV, we remove all the leaf nodes that do not belong to $\{e_1..e_{|Qd|}\}$. We do this iteratively until no leaf node is removed. The result is a minimum tree that contains nodes $e_1, \ldots, e_{|Qd|}$ in LV

ALGORITHM 3. LD-Gen(GS, Q, S, DII)

Input: The label-graph GS, a query Q =(Qd,Qg, α, β), the dimensional inverted index DII and the set of LVs, S.
Output: The set of all LDs.

```
1: Q  ← Φ;
2: for each keyword di in Qd do
3:    for each e Є list(di) do
4:        for each LV  Є S do
5:            if e.Rel Є LV then
6:               LV.listi ←LV.listi U {e};
7: for each LV Є S do.
8:        if LV.listi ≠ Φ  for any 1≤ i ≤|Qd|.  then
9:            for all (e1 . . . , e|Qd| ) Є LV.list1X…….XLV.list|Qd|
          do
10:              construct a minimal LD that only contains
                  attribute-nodes (e1,. . . ,e|Qd| );
11: Q ← Q U { LD};
12: return Q;
```

V. EVALUATE ALL LDS

Here , for a structural statistics keyword query Q=(Qd,Qg, α, β), we give a two-phase approach to compute structural statistics for all the groups under a given LD.

A. Two-Phase Approach

We propose a new two-phase approach, namely, bottom-up followed by top-down, after pruning unnecessary nodes/ edges from GD that need to evaluate an LD(R). For a given LD(R) to be evaluated, let LV(R) be the labelled LV that generates LD(R). In the bottom-up phase, we collect statistics for TF-IDF, namely, $tf_w(T)$ and idf_w using general-keywords. The statistics collection is done in a bottom-up fashion, and will finish when it finally evaluates the relation R which is the root of LD(R). Let R' be the set of tuples in R that contain statistics for at least one general-keyword in Qg. At the end of this phase, we have a set of {Gtree(t_γ)} for t_γ in R', and we compute all α(Gtree(t_γ)). In the top-down phase, we start from those tuples t_γ in R'.We retrieve all leaf attribute nodes that t_γ can reach using the depth first search from t_γ. If the leaf attribute nodes contain all the required dimensional keywords in Q_d, then t_γ has a valid Dtree(t_γ). At the end of this top-down process, we compute γ for all such t_γ in R' that has a valid Dtree(t_γ).

We first discuss the pruning process using dimensional keywords before the two-phase approach. Given adimensional keyword d_i, a node v in G_D can be pruned by d_i if there does not exist a Dtree that contains both keyword di and node v. We call v an unnecessary node if v can be pruned by any dimensional keyword in the query. Given an LD, we consider whether a keyword d_i Є Qd has enough pruning power using Pow(di,LD) which is computed as follows[5]: suppose X(A,B) is the join selectivity in relation A that can join a tuple in relation B, i.e., the fraction of tuples of A that have a matching tuple in B. Suppose the attribute in relation Ri in LD contains di, and assume Pi is the path from the root of LD to Ri. We have

$$Pow(d_i) = \frac{1}{R_i.c(d_i).\prod_{(A,B)\in P_i} X(A,B)} \quad (2)$$

Here $R_i.c(d_i)$ is the number of tuples in R_i that contain the keyword d_i in the specified attribute of R_i in LD. The basic idea behind is as follows: for any join sequence A ⋈ B ⋈ C, we assume that for any tuples a Є A, b Є B,and c Є C, whether a can be joined with b and whether b can be joined with c are independent to each other. $R_i.c(d_i).\prod_{(A,B)\in P_i} X(A,B)$ is the expected number of root nodes to reach a node with keyword di. Intuitively, a keyword that ends up with a smaller expected number of the root nodes has higher pruning power.

For any general-keyword $g_j \in Qg$, suppose for any text-attribute A_i, $R(A_i)$ is the relation of A_i, and $P(A_i)$ is the path from the root of LD to the relation $R(A_i)$.We compute the pruning power of a general-keyword as follows[5]:

$$Pow(g_i,LD) = \frac{1}{\sum_{Ai \in LD} R(A_i).c(g_i).\prod_{(A,B) \in P(Ai)} x(A,B)} \quad (3)$$

Based on the equations, we decide whether a dimensional keyword di $\in Q_d$ has enough power to prune, or in other words, it is cost-effective to reduce G_D. We sort all dimensional-keywords di Є Qd in decreasing order . If the pruning power of di is larger than the largest pruning power of all general-keywords in Qg, then we use di to reduce G_D by removing all the tuple-nodes that cannot reach any attribute nodes containing di.

The algorithm is shown in Algorithm 4[5]. We assume that the structure of the data graph is held in memory. First, we prune unnecessary nodes from G_D using Qd if they have enough pruning power (lines 5-7). Second, in the bottom-up phase, we compute trees in a sense to collect all needed information to compute α for every Gtree using Q_g. It is done from the leaf toward the root which is a tuple in r(R) for the LV(R) that generates the given LD (lines 9-25).Finally, in a top-down phase, we aggregate for each group based on Qd (lines 27-31).

ALGORITHM 4. LD-Eval(Q, LD, GD)
Input: A query Q ={Qd,Qg, α, β }, LD
and a database graph GD(V ,E).
Output: Aggregates for all groups

1: Γ ← Φ;
2: let LV(R) be the LV that generates LD;
3: for all relation node P Є LV(R) do
4: P.set ← Φ;
5: for all d_i Є Q_d sorted by decreasing order of Pow(d_i,LD) do
6: if Pow(d_i,LD) > $_{max_{gi} \in Qg}$ (Pow(g_i,LD)) then
GD ← prune(G_D,d_i,LD);
8: // The bottom-up phase
9: for all relation node P Є LV(R) in the order from leaves to the root **do**
10: for i=1 to |Q_g| do
11: for all tuple-node t_P Є P.contain(g_i) **do**
12: t_P .cnt_i ← t_P .cnt_i + t_p.count(gi)
13: P.has_i ← P.has_i U {t_P};
14: P.set ← P.set U {t_P};
15: for all child of P, C, in LV(R) do
16: for all node tC Є C.set do
17: for all node t_P Є P such that $t_P \rightarrow t_C$ Є E(GD) do
18: P.set ← P.set U {t_P};
19: for i= 1 to |Q_g| do
20 $t_{p.}cnt_i$ ← t_P :cnt_i + t_c.cnt_i;
21: if t_C.cnt_i > 0 then P.has_i P.has_i ← P.has_i U {t_p}
22: for all tuple-node t Є R.set do
23: for i = 1 to |Qg| do
24: tf_{gi} (t) ← t.cnt_i; df_{gi} (LV(R) ←|R:has_i|;
25: t.score ← α(t) using all tf_{gi}(t) and df_{gi}(LV(R));
26: // The top-down phase
27: for all t Є R.set do
28: let ai be the attribute value in the attribute
A_i Є att(LD) that contains d_i, for all d_i Є Q_d;
29: let γ be a group represented by (a1, a2,.. $a_{|Qd|}$)
30: γ. score = β(γ. Score, t.score);
31: Γ ← Γ U {γ} if γ not in Γ;
32: return Γ

VI. CONCLUSIONS

In this paper, we studied how to compute structural statistics for all groups of tuples in RDB using the keyword query. By using a new two-step approach, we compute all label-trees for a structural statistics keyword query, and a new two-phase algorithm used to compute group-&-aggregate using the label-trees computed.

REFERENCES

[1] S. Agrawal, S. Chaudhuri, and G. Das, “DBXplorer: A System for Keyword-Based Search over Relational Databases,” Proc. 18th Int’l Conf. Data Eng. (ICDE ’02), 2002.

[2] V. Hristidis, L. Gravano, and Y. Papakonstantinou, “Efficient IRStyle Keyword Search over Relational Databases,” Proc. 29th Int’l Conf. Very Large Data Bases (VLDB ’03), 2003

[3] V. Hristidis and Y. Papakonstantinou, “DISCOVER: Keyword Search in Relational Databases,” Proc. 28th Int’l Conf. Very Large Data Bases (VLDB ’02), 2002.

[4] Y. Luo, X. Lin, W. Wang, and X. Zhou, “Spark: Top-K Keyword Query in Relational Databases,” Proc. ACM SIGMOD Int’l Conf. Management of Data (SIGMOD ’07), 2007

[5] Lu Qin,Jeffrey Xu Yu, and Lijun Chang ,“Computing Structural Statistics by Keywords in Data bases”,IEEE transactions on knowledge And Data Engg.,oct 2012

[6] L. Qin, J.X. Yu, and L. Chang, “Computing Structural Statistics by Keyword in Databases,” Proc. IEEE 27th Int’l Conf. Data Eng. (ICDE ’11), 2011.

[7] L. Qin, J.X. Yu, and L. Chang, “Keyword Search in Databases: The Power of Rdbms,” Proc. ACM SIGMOD Int’l Conf. Management of Data (SIGMOD ’09), 2009

Ambily Jose received the B.Tech. degree in Computer Science and Engineering from Cochin University of Science and Technology, Cochin, Kerala. At present persuing the Master of Technology in Computer Science and Engineering Department at BTL institute of Technology, Bangalore.

Jobin George currently working as a Senior Software Engineer in Wipro Technologies,Bangalore, India

Liliya T Jose currently working as an Assistant Professor in Vimal Jyothy Engg.College affiliated to Kannur University, Kerala

