Implementation of Skyline Sweeping Algorithm

B. R. Ambedkar Kota, G. Jayaraju
1,2 Dept. of CSE, Lenora College of Engineering, Rampachodavaram, AP, India

Abstract

Searching keywords in databases is complex task than search in files. Information Retrieval (IR) process search keywords from text files and it is very important that queering keyword to the relational databases. Generally to retrieve data from relational database Structure Query Language(SQL) can be used to find relevant records from the database. There is natural demand for relation database to support effective and efficient IR Style Keyword queries. This paper describes problem of supporting effective and efficient top-k keyword search in relational databases also describe the frame which takes keywords and K as inputs and generates top-k relevant records. The results of implemented system with Skyline Sweeping(S.S) Algorithm shows that it is one effective and efficient style of keyword search.

Keywords

Top-K, Keyword Search, Relational Database, Information Retrieval

I. Introduction

Internet search engines have popularized keyword based search. Users submit keywords to the search engine and a ranked list of documents is returned to the user. A significant amount of the world's enterprise data resides in relational databases. It is important that users be able to seamlessly search and browse information stored in these databases as well. Searching databases on the internet and intranet today is primarily enabled by customized web applications closely tied to the schema of the underlying databases, allowing users to direct searches in a structured manner. Examples of such searches within, say a bookseller's database may be “Books → Travel → Lonely Planet → Asia”, or “Books → Travel → Rough Guides → Europe”. With the growth of the World Wide Web, there has been a rapid increase in the number of users who need to access online databases without having a detailed knowledge of schema or query languages; even relatively simple query languages designed for non-experts are too complicated for such users. Increasing amount of text data stored in relational databases, there is a demand for RDBMS to support keyword queries over text data. As a search result is often assembled from multiple relational tables, traditional IR-style ranking and query evaluation methods cannot be applied directly. This paper, describes the effectiveness and the efficiency issues of answering top-k keyword query in relational database systems. We propose a new ranking formula by adapting existing IR techniques based on a natural notion of virtual document. The rest of the paper is organized as follows: Section 2 Related work. Section 3 presents Problem Description Section 4 Frame works and algorithms optimized for efficient top-k retrieval. Experimental results are reported in Section 5. Section 6 concludes the paper.

II. Related work

The FFF search mechanism at the websites that provides facts and figures may be augmented by DBXplorer technology. DataSpot is a commercial system that supports keyword-based searches by extracting the content of the database into a hyperbase. Thus, this approach duplicates the content of the database, which makes data integrity and maintenance difficult. Microsoft's English Query provides a natural language interface to a SQL database. DISCOVER has proposed a breadth-first CN enumeration algorithm that is both sound and complete. The algorithm is essentially enumerating all subgraphs of size k that does not violate any pruning rules. The algorithm varies k from 1 to some search range threshold M. Three pruning rules are used and they are listed below. Issue an SQL query for each CN and union them to find the top-k results by their relevance scores. DISCOVER2 introduce two alternative query evaluation strategies: sparse and global pipeline algorithms, both optimized for stopping the query execution immediately after the true top-k-th result can be determined.

II. System Description

Consider a relational schema R as a set of relations \{R_1, R_2, \ldots, R|R|\}. These relations are interconnected at the schema level via foreign key to primary key references and denote \(R_i \rightarrow R_j\) if \(R_i\) has a set of foreign key attribute(s) referencing \(R_j\)'s primary key attribute(s), following the convention in drawing relational schema graphs. For simplicity, we assume all primary key and foreign key attributes are made of single attribute, and there is at most one foreign key to primary key relationship between any two relations and do not impose such limitations in our implementation. A query Q consists of (1) a set of distinct keywords, i.e., \(Q = \{w_1, \ldots, w_i\}\); and (2) a parameter k indicating that a user is only interested in top-k results ranked by relevance scores associated with each result.
III. Frame Work and Algorithm

Skyline Sweeping algorithm designed to minimize the number of join checking operations, which typically dominates the cost of the algorithm. This intuition is that if there are two candidates x and y and the upper bound score of x is higher than that of y, y should not be checked unless x has been checked. Therefore, we should arrange all the candidates to be checked according to their upper bound scores. A naïve strategy is to calculate the upper bound scores for all the candidates, sort them according to the ascending order, and check them one by one according to this optimal order. This will incur excessive amount of unnecessary work, since not all the candidates need to be checked.

Algorithm: Skyline Sweeping Algorithm

1: \(Q.\text{push}(\langle m \ z \ | \ | \ 1, 1, \ldots, 1 \rangle, \text{calc uscore}(\langle m \ z \ | \ | \ 1, 1, \ldots, 1 \rangle))\)
2: \(\text{top-k} \leftarrow 0\)
3: while \(\text{top-k}[\text{top-k}].\text{score} < Q.\text{head().uscore}\) do
4: \(\text{head} \leftarrow Q.\text{pop max()}\)
5: \(r \leftarrow \text{executeSQL(formQuery(head))}\)
6: if \(r \neq \text{nil}\) then
7: \(\text{top-k}.\text{insert}(r, \text{score}(r))\)
8: for \(i \leftarrow 1\) to \(m\) do
9: \(t \leftarrow \text{head.dup()}\)
10: \(t.i \leftarrow t.i + 1\)
11: \(Q.\text{push}(t, \text{calc uscore}(t))\) {According to Equation (4)}
12: if \(t.i > 1\) then
13: break
14: return \(\text{top-k}\)

A result list, \(\text{top-k}\), contains no more than \(k\) results ordered by the descending real scores. The main data structure is a priority queue, \(Q\), containing all the candidates (which are mapped to multi-dimensional points) according to the descending order of their upper bound scores. The algorithm also maintains the invariant that the candidate at the head of the priority queue has the highest upper bound score among all candidates in the CN. The invariant is maintained by (a) pushing the candidate formed by the top tuple from all dimensions into the queue (Line 1), and (b) whenever a candidate is popped from the queue, its adjacent candidates are pushed into the queue together with their upper bounds (Lines 8–13). The algorithm stops when the real score of the current top-k-th result is no smaller than the upper bound score of the head element of the priority queue; the latter is exactly the upper bound score of all the unprocessed candidates.

Fig. 1:

A. Rank Function

\[
\text{score}(T, Q) = \sum_{t \in T} \text{score}(t, Q)
\]
\[
\text{score}(t, Q) = \sum_{w \in t \cap Q} \frac{1 + \ln(1 + (\text{tfw}(t)))}{(1 - s) \cdot \frac{\text{dlit}}{\text{avdlit}}} \ln(\text{idf}_w),
\]

where \(\text{idf}_w = \frac{N_{\text{Rel}(t)} + 1}{df_w(\text{Rel}(t))}\).

\(\text{tfw}(t)\) denotes the number of times a keyword \(w\) appears in a database tuple \(t\), \(\text{dlit}\) denotes the length of the text attribute of a tuple \(t\), and \(\text{avdlit}\) is the average length of the text attribute in the relation which \(t\) belongs to (i.e., \(\text{Rel}(t)\)). \(N_{\text{Rel}(t)}\) denotes the number of tuples in \(\text{Rel}(t)\), and \(df_w(\text{Rel}(t))\) denotes the number of tuples in \(\text{Rel}(t)\) that contain keyword \(w\). The score of a JTT is the sum of the local scores of every tuple in the JTT.

Fig. 2:

IV. Results

The below diagrams represent keyword search results normal as well as top-3 key word searching hanks 2001.


Table 1: Top-3 Search Result on Our System

<table>
<thead>
<tr>
<th>JTT</th>
<th>CN</th>
<th>tfactor</th>
<th>tdimension</th>
<th>Score</th>
<th>Our Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>c3 → p2</td>
<td>C</td>
<td>0.1</td>
<td>0.1</td>
<td>2.0</td>
<td>3.0</td>
</tr>
<tr>
<td>c1 → p1</td>
<td>C → PQ</td>
<td>0.1</td>
<td>0.1</td>
<td>1.0</td>
<td>2.0</td>
</tr>
<tr>
<td>c2 → p2</td>
<td>C → PQ</td>
<td>0.1</td>
<td>0.1</td>
<td>1.0</td>
<td>2.0</td>
</tr>
</tbody>
</table>

Running example (shown in Figures above). In the example, \(R = \{P, C, U\}\).1 Foreign key to primary key relationships are: \(C \rightarrow P\) and \(C \rightarrow U\). A user wants to retrieve top-3 answer to the query “maxtor netvista”. Some example JTTs include: \(c3 \rightarrow p2\), \(c1 \rightarrow p1\), \(c2 \rightarrow p2\), and \(c2 \rightarrow p2 ← c3\). The first JTT belongs to CN CQ; the next three JTTs belong to CN CQ → PQ; and the last JTT belongs to CN CQ → PQ ← CQ. Note that \(c3 → u3\) is not a valid JTT to the query, as the leaf node \(u3\) does not contribute to a match to the query. A possible answer for this top-3 query may be: \(c3 \rightarrow p2\), \(c3 \rightarrow p2\), and \(c1 \rightarrow p1\). We believe that most users will prefer \(c1 \rightarrow p1\) to \(c2 \rightarrow p2\), because the former complaint is really about an IBM Netvista equipped with a Maxtor disk, and
that it is not certain whether Product p2 mentioned in the latter JTT is equipped with a Maxtor hard disk or not.

VI. Conclusion
This paper, studied supporting effective and efficient top-k keyword queries over relational databases, and proposed a new ranking method that adapts the state-of-the-art IR ranking function and principles into ranking trees of joined database tuples. The ranking method also has several salient features over existing ones. We also studied query processing method tailored for our non-monotonic ranking functions. Two algorithms were proposed that aggressively minimize database probes. We have conducted extensive experiments on large-scale real databases. The experimental results confirmed that our ranking method could achieve high precision with high efficiency to scale to databases with tens of millions of tuples.

References