Enhanced IR Schemes for Summarizing for Relational Databases

[1]K.Chandra Mouli, [2]G.Aparna
[1,2]Aditya Engineering College, Surampalem, East-Godavari, AP, India

Abstract
Commercial relational database management systems (RDBMSs) generally provide querying capabilities for text attributes that incorporate state-of-the-art information retrieval (IR) relevance ranking strategies, but this search functionality requires that queries specify the exact column or columns against which a given list of keywords is to be matched. This requirement can be cumbersome and inflexible from a user perspective: good answers to a keyword query might need to be "assembled" –in perhaps unforeseen ways– by joining tuples from multiple relations. This observation has motivated recent research on free-form keyword search over RDBMSs. In this paper, we adapt IR-style document-relevance ranking strategies to the problem of processing free-form keyword queries over RDBMSs. Our query model can handle queries with both AND and OR semantics, and exploits the sophisticated single-column text-search functionality often available in commercial RDBMSs. We develop query-processing strategies that build on a crucial characteristic of IR-style keyword search: only the few most relevant matches –according to some definition of "relevance"– are generally of interest. Consequently, rather than computing all matches for a keyword query, which leads to inefficient executions, our techniques focus on the top-k matches for the query, for moderate values of k.

Keywords
Top-k, Keyword Search, Relational Database, Information Retrieval

I. Introduction

The integration of DB and IR provides flexible ways for users to query information in the same platform. On one hand, the sophisticated DB facilities provided by RDBMSs assist users to query well-structured information using SQL. On the other hand, IR techniques allow users to search unstructured information using keywords based on scoring and ranking, and do not need users to understand any database schemas.

Commercial RDBMSs generally provide querying capabilities for text attributes that incorporate state-of-theart information retrieval (IR) relevance ranking strategies. This search functionality requires that queries specify the exact column or columns against which a given list of keywords is to be matched. For example, a query:

SELECT * FROM Complaints C WHERE CONTAINS (C.comments, 'disk crash', 1) > 0 ORDER BY score(1) DESC

on Oracle 9.1 1 returns the rows of the Complaints table that match the keyword query [disk crash], sorted by their score as determined by an IR relevance-ranking algorithm. Intuitively, the score of a tuple measures how well its comments field matches the query [disk crash]. The requirement that queries specify the exact columns to match can be cumbersome and inflexible from a user perspective: good answers to a keyword query might need to be "assembled" –in perhaps unforeseenways– by joining tuples from multiple relations:

Free-form keyword search over RDBMSs has attracted recent research interest. Given a keyword query, systems such as DBXplorer [1] and DISCOVER [11] join tuples from multiple relations in the database to identify tuple trees with all the query keywords ("AND" semantics). Unfortunately, these techniques do not consider IR-style ranking heuristics that have proved effective over text.

A key contribution of this paper is the incorporation of IR-style relevance ranking of tuple trees into our query processing framework. In particular, our scheme fully exploits single-attribute relevance-ranking results if the RDBMS of choice has text-indexing capabilities (e.g., as is the case for Oracle 9.1, as discussed above). By leveraging state-of-theart IR relevance-ranking functionality already present in modern RDBMSs, we are able to produce high quality results for free-form keyword queries. For example, a query [disk crash on a netvista] would still match the comments attribute of the first Complaints tuple above with a high relevance score, after word stemming (so that "crash" matches "crashed") and stop-word elimination (so that the absence of "a" is not weighed too highly). Our scheme relies on the IR engines of RDBMSs to perform such relevance-ranking at the attribute level, and handles both AND and OR semantics.

Unfortunately, existing query-processing strategies for keyword search over RDBMSs are inherently inefficient, since they attempt to capture all tuple trees with all query keywords. Thus these strategies do not exploit a crucial characteristic of IR-style keyword search, namely that only the top 10 or 20 most relevant matches for a keyword query –according to some definition of "relevance"– are generally of interest. The second contribution of this paper is the presentation of efficient query processing techniques for our IR-style queries over RDBMSs that heavily exploit this observation. As we will see, our techniques produce the top-k matches for a query –for moderate values of k– in a fraction of the time taken by state-of-the-art strategies to compute all query matches. Furthermore, our techniques are pipelined, in the sense that execution can efficiently resume to compute the "next-k" matches if the user so desires.

II Related Work

Extracting ranking functions has been extensively investigated in areas outside database research such as Information Retrieval. The vector space model as well as probabilistic information retrieval (PIR) models and statistical language models are very successful in practice. While our approach has been inspired by PIR models, we have adapted and extended them in ways unique to our situation, e.g., by leveraging the structure as well as correlations present in the structured data and the database workload.

In database research, there has been some work on ranked retrieval from a database. The early work considered vague/imprecise similarity-based querying of databases. The problem of integrating databases and information retrieval systems has been attempted in several works. Information retrieval based approaches have been extended to XML retrieval. Keyword-query based retrieval systems over databases have been proposed invarious papers. SQL extensions in which users can specify ranking functions via soft constraints in the form of preferences. The distinguishing aspect of our work from the above is that we espouse automatic extraction of PIR-based ranking functions through data and

workload statistics.

DBXplorer and DISCOVER exploit the RDBMS schema, which leads to relatively efficient algorithms for answering keyword queries because the structural constraints expressed in the schema are helpful for query processing. These two systems rely on a similar architecture, Unlike DBXplorer and DISCOVER, our techniques are not limited to Boolean-AND semantics for queries, and we can handle queries with both AND and OR semantics. In contrast, DBXplorer and DISCOVER (as well as BANKS) require that all query keywords appear in the tree of nodes or tuples that are returned as the answer to a query. Furthermore, we employ ranking techniques developed by the IR community, instead of ranking answers solely based on the size of the result as in DBXplorer and DISCOVER. Also, our techniques improve on previous work in terms of efficiency by exploiting the fact that free-form keyword queries can generally be answered with just the few most relevant matches. Our work then produces the "top-k" matches for a query fast, for moderate values of k.

III Framework

In this section, we specify the query model (Section A), together with the family of scoring functions that we consider to identify the top-k answers for a query (Section B).

A. Query Model

Consider a database with n relations $R_1, \ldots, R_n$. Each relation Ri has mi attributes $a_i, \ldots, a_i m_i$, a primary key and possibly foreign keys into other relations. The schema graph G is a directed graph that captures the foreign key relationships in the schema. G has a node for each relation Ri, and an edge Ri → Rj for each primary key to foreign key relationship from Ri into Rj . Fig. 1, shows the schema graph of our Complaints database running example, while fig. 2, shows a possible instance of this database. A top-k keyword query is a list of keywords Q = [w1, . . ., wm]. The result for such a top-k query is a list of the k joining trees of tuples T whose Score(T,Q) score for the query is highest, where Score(T,Q) is discussed below. The query result is sorted in descending order of the scores. We require that any joining tree T in a query result be minimal: if a tuple t with zero score is removed from T , then the tuples remaining in T are "disconnected" and do not form a joining tree. In other words, T cannot have a leaf tuple with zero score. As an example, for a choice of ranking

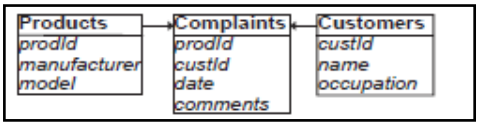

Fig. 1: Schema of the Complaints Database

function Score the results for a top-3 query [Netvista Maxtor] over our Complaints database could be (1) c3; (2) p2 → c3; and (3) p1 → c1. Finally, we do not allow any tuple to appear more than once in a joining tree of tuples.

Complaints

tupleId	prodId	custId	date	comments
c_1	p121	c3232	6-30-2002	"disk crashed after just one week of moderate use on an IBM Netvista X41"
c_2	p131	c3131	7-3-2002	"lower-end IBM Netvista caught fire, starting apparently with disk"
c_3	p131	c3143	8-3-2002	"IBM Netvista unstable with Maxtor HD"

Products

tupleId	prodId	manufacturer	model
p_1	p121	"Maxtor"	"D540X"
p_2	p131	"IBM"	"Netvista"
p_3	p141	"Tripplite"	"Smart 700VA"

Customers

tupleId	custId	name	occupation
u_1	c3232	"John Smith"	"Software Engineer"
u_2	c3131	"Jack Lucas"	"Architect"
u_3	c3143	"John Mayer"	"Student"

Fig. 2: An Instance of the Complaints Database

B. Ranking Functions

We now discuss how to rank joining trees of tuples for a given query. Result ranking has been addressed by other keyword-search systems for relational data. The approaches DISCOVER and DBXplorer capture the size and "structure" of a query result in the score that it is assigned, but do not leverage further the relevance-ranking strategies developed by the IR community over the years. These strategies –which were developed exactly to improve document-ranking quality for free-form keyword queries– can naturally help improve the quality of keyword query results over RDBMSs. Furthermore, modern RDBMSs already include IR-style relevance ranking functionality over individual text attributes, which we exploit to define our ranking scheme. Specifically, the score that we assign to a joining tree of tuples T for a query Q relies on:

• Single-attribute IR-style relevance scores Score(ai,Q) for each textual attribute ai □ T

and query Q, as determined by an IR engine at the RDBMS, and

• A function Combine, which combines the singleattribute scores into a final score for T .

In addition to the combining function, queries should specify whether they have Boolean AND or OR semantics. The AND semantics assigns a score of 0 to any tuple tree that does not include all query keywords, while tuple trees with all query keywords receive the score determined by Combine. In contrast, the OR semantics always assigns a tuple tree its score as determined by Combine, whether the tuple tree includes all query keywords or not.

IV. System Architecture

The architecture of our query processing system relies whenever possible on existing, unmodified RDBMS components. Specifically, our architecture (fig. 3) consists of the following modules:

A. IR Engine

As discussed, modern RDBMSs include IR-style text indexing functionality at the attribute level. The IR Engine module of our architecture exploits this functionality to identify all database tuples that have a non-zero score for a given query. The IR Engine relies on the IR Index, which is an inverted index that associates each keyword that appears in the database with a list of occurrences of the keyword; each occurrence of a keyword is recorded as a tuple

attribute pair. Our implementation uses Oracle Text, which keeps a separate index for each relation attribute. We combine these individual indexes to build the IR Index. When a queryQ arrives, the IR Engine uses the IR Index to extract from each relation R the tuple set RQ = {t □ R | Score(t,Q) > 0}, which consists of the tuples of R with a non-zero score for Q. The tuples t in the tuple sets are ranked in descending order of Score(t,Q), as required by the top-k query processing algorithms

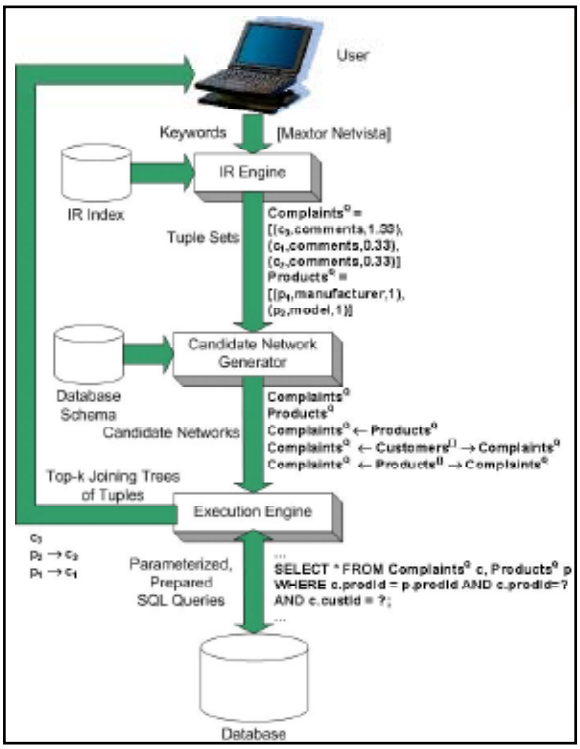

Fig. 3: Architecture of Our Query Processing System

B. Candidate Network Generator

The next module in the pipeline is the Candidate Network (CN) Generator, which receives as input the non-emptytuple sets from the IR Engine, together with the database schema and a parameterM that we explain below. The key role of this module is to produce CNs, which are join expressions to be used to create joining trees of tuples that will be considered as potential answers to the query. Specifically, a CN is a join expression that involves tuple sets plus perhaps additional "base" database relations. We refer to a base relationR that appears in a CN as a free tuple set and denote it as R{}. Intuitively, the free tuple sets in a CN do not have occurrences of the query keywords, but help "connect" (via foreign-key joins) the (non-free) tuple sets that do have non-zero scores for the query. Each result T of a CN is thus a potential result of the keyword query.

We say that a joining tree of tuples T belongs to a CN C (T ϵ C) if there is a tree isomorphism mapping h from the tuples of T to the tuple sets of C. For example, in fig. 2, (c1 ← p1) □ (ComplaintsQ ← ProductsQ). The input parameterM bounds the size (in number of tuple sets, free or non-free) of the CNs that this module produces. The notion of CN was introduced in DBXplorer and DISCOVER. As discussed, DISCOVER and DBXplorer require that each joining tree of tuples in the query answer contain all query keywords. To produce all answers for a query with this AND semantics, these systems create multiple tuple sets for each database relation. Specifically, a separate tuple set is created for each combination of keywords in Q and each relation. This generally leads to a number of CNs that is exponential in the query size, which makes query execution prohibitively expensive for queries of more than a very small number of keywords or for values of M greater than 4 or so.

In contrast, we only create a single tuple set RQ for each relation R, as specified above. For queries with AND semantics, a postprocessing step checks that we only return tuple trees containing all query keywords. As we will see, this characteristic of our system results in significantlyfaster executions, which in turn allows us to handle larger queries and also consider larger CNs. The CN generation algorithm is based on that of the DISCOVER system, and is not explained here in full detail due to lack of space. Conceptually, we first create the tuple set graph from the database schema graph and the tuple sets returned by the IR Engine module. We progressively expand each CN s ϵ S by adding a tuple set adjacent to s in the tuple set graph. We consider s to be a CN and hence part of the output of this module if it satisfies the following properties:

1. The number of non-free tuple sets in s does not exceed the number of query keywordsm: This constraint guarantees that we generate a minimum number of CNs while not missing any result that contains all the keywords, which is crucial for Boolean-AND semantics. That is, for every result T that contains every keyword exactly once, a CN C exists such that T ϵ C.
2. No leaf tuple sets of s are free: This constraint ensures CN "minimality
3. s does not contain a construct of the form R → S ← R: If such a construct existed, every resulting joining tree of tuples would contain the same tuple more than once.

The size of a CN is its number of tuple sets. All CNs of size 3 or lower for the query [Maxtor Netvista] are shown in fig. 3.

C. Execution Engine

The final module in the pipeline is the Execution Engine, which receives as input a set of CNs together with the non-free tuple sets. The Execution Engine contacts the RDBMS's query execution engine repeatedly to identify the top-k query results. The Execution Engine module is the most challenging to implement efficiently.

V. Execution Algorithms

We now present algorithms for a core operation in our system: given a set of CNs together with a set of non-free tuple sets, the Execution Engine needs to efficiently identify the top-k joining trees of tuples that can be derived . First, we describe the Naive algorithm, a simple adaptation of query processing algorithms used in prior work. Second, we present the Sparse algorithm, which improves on the Naive algorithm by dynamically pruning some CNs during query evaluation. Third, we describe the Single Pipelined algorithm, which calculates the top-k results for a single CN in a pipelined way. Fourth, we present the Global Pipelined algorithm, which generalizes the Single Pipelined algorithm to multiple CNs and can then be used to calculate the final result for top-k queries. Finally, we introduce the Hybrid algorithm, which combines the virtues of both the Global Pipelined and the Sparse algorithms.

A. Naive Algorithm

The Naive algorithm issues a SQL query for each CN for a top-k query. The results from each CN are combined in a sort-merge manner to identify the final top-k results of the query. This approach is an adaptation of the execution algorithms of prior work for keyword-search queries. As a simple optimization in our experiments, we only get the top-k results from each CN

ISSN : 0976-8491 (Online) | ISSN : 2229-4333 (Print)

according to the scoring function, and we enable the top-k "hint" functionality, available in the Oracle 9.1 RDBMS. In the case of Boolean-AND semantics, the Naive algorithm (as well as the Sparse algorithm presented below) involves an additional filtering step on the stream of results to check for the presence of all keywords.

B. Sparse Algorithm

The Naive algorithm exhaustively processes every CN associated with a query. We can improve query-processing performance by discarding at any point in time any (unprocessed) CN that is guaranteed not to produce a top-k match for the query. Specifically, the Sparse algorithm computes a bound MPSi on the maximum possible score of a tuple tree derived from a CN Ci. If MPSi does not exceed the actual score of k already produced tuple trees, then CN Ci can be safely removed from further consideration. To calculate MPSi, we apply the combining function to the top tuples (due to the monotonicity property in Definition 2) of the non-free tuple sets of Ci. That is, MPSi is the score of a hypothetical joining tree of tuples T that contains the top tuples from every non-free tuple set in Ci. As a further optimization, the CNs for a query are evaluated in ascending size order. This way, the smallest CNs, which are the least expensive to process and are the most likely to produce high-score tuple trees using the combining function above, are evaluated first.

C. Single Pipelined Algorithm

The Single Pipelined algorithm receives as input a candidate network C and the non-free tuple sets TS1, . . . , TSv that participate in C. Recall that each of these non-free tuple sets corresponds to one relation, and contains the tuples in the relation with a non-zero match for the query. Furthermore, the tuples in TS i are sorted in descending order of their Score for the query. The output of the Single Pipelined Algorithm consists of a stream of joining trees of tuples T in descending Score(T,Q) order. In effect, the Single Pipelined algorithm can start producing results before examining the entire tuple sets. For this, we maintain an effective estimate of the Maximum Possible Future Score (MPFS) that any unseen result can achieve, given the information already gathered by the algorithm.

D. Global Pipelined Algorithm

The Global Pipelined algorithm builds on the Single Pipelined algorithm to efficiently answer a top-k keyword query over multiple CNs. The algorithm receives as input a set of candidate networks, together with their associated non-free tuple sets, and produces as output a stream of joining trees of tuples ranked by their overall score for the query. The key idea of the algorithm is the following. All CNs of the keyword query are evaluated concurrently following an adaptation of a priority preemptive, round robin protocol , where the execution of each CN corresponds to a process. Each CN is evaluated using a modification of the Single Pipelined algorithm, with the "priority" of a process being the MPFS value of its associated CN.

E. Hybrid Algorithm

The Hybrid algorithm critically relies on the accuracy of the result-size estimator. For queries with OR semantics, we can simply rely on the RDBMS's result-size estimates, which we have found to be reliable. In contrast, this estimation is more challenging for queries with AND semantics: the RDBMS that we used for our implementation, Oracle 9i, ignores the text index when producing estimates. Therefore, we can obtain from the RDBMS an estimate S of the number of tuples derived from a CN (i.e., the number of tuples that match the associated join conditions), but we need to adjust this estimate so that we consider only tuple trees that contain all query keywords.

VI. Perforamnce

The parameters that we vary in the experiments are (a) the maximum sizeM of the CNs, (b) the number of results k requested in top-k queries. We compared the following algorithms:

- The Naive algorithm.
- The Sparse algorithm.
- The Single Pipelined algorithm (SA).
- The Global Pipelined algorithm (GA).
- SASymmetric and GASymmetric are modifications of SA and GA, respectively.
- The Hybrid algorithm.

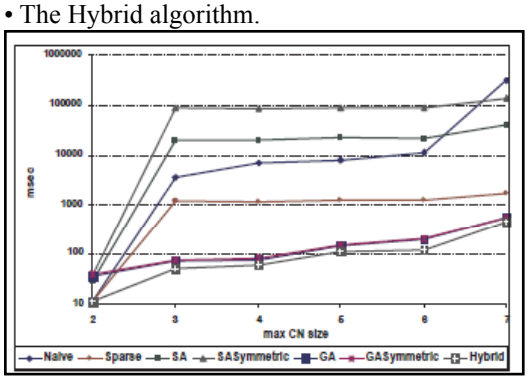

Fig. 4: OR Semantics: Effect of the Maximum Allowed CN Size

A. Boolean-OR Semantics

Effect of the maximum allowed CN size: Fig. shows the average query execution time over 100 two-keyword top-10 queries, where each keyword is selected randomly from the set of keywords in the DBLP database. GA, GASymmetric, and Hybrid are orders of magnitude faster than the other approaches.

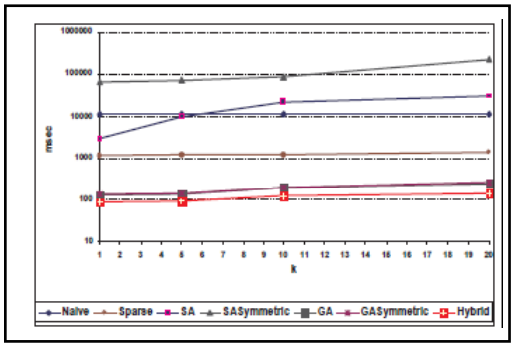

Fig. 5: OR Semantics: Effect of the Number of Objects Requested, k

B. Effect of the Number of Objects Requested

The average execution times over 100 queries are shown in fig. 5. Notice that the performance of Naive remains practically unchanged across different values of k, in contrast to the pipelined algorithms whose execution time increases smoothly with k.

C. Boolean-AND Semantics

Effect of M, k: Below figures shows That Hybrid performs almost identically as Sparse: for AND semantics, the number of potential query results containing all the query keywords is relatively small, so Hybrid selects Sparse for almost all queries.

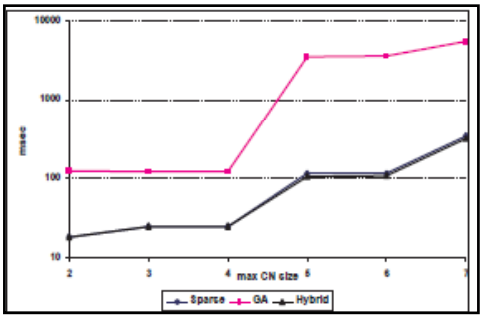

Fig. 6: AND Semantics: Effect of the Maximum Allowed CN Size

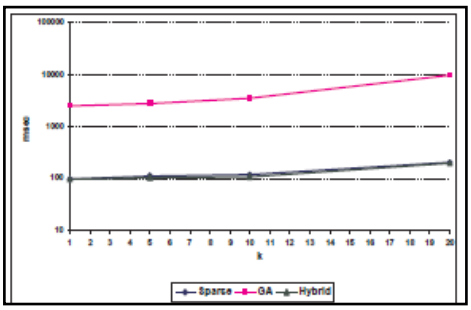

Fig. 7: AND Semantics: Effect of the Number of Objects Requested, k

VII. Conclusion

In this paper, we presented a system for efficient IR-style keyword search over relational databases. A query in our model is simply a list of keywords, and does not need to specify any relation or attribute names. The answer to such a query consists of a rank of "tuple trees," which potentially include tuples from multiple relations that are combined via joins. To rank tuple trees, we introduced a ranking function that leverages and extends the ability of modern relational database systems to provide keyword search on individual text attributes and rank tuples accordingly. In particular, our ranking function appropriately combines the RDBMS provided scores of individual attributes and tuples. As another contribution of the paper, we introduced several top-k query-processing algorithms whose relative strengths depend, for example, on whether queries have Boolean-AND or OR semantics. We also presented a "hybrid" algorithm that decides at run-time the best strategy to follow for a given query, based on result-size estimates for the query. This hybrid algorithm has the best overall performance for both AND and OR query semantics.

References

[1] S. Agrawal, S. Chaudhuri, G. Das,"DBXplorer: A system for keyword-based search over relational databases", In ICDE, 2002.

[2] G. Bhalotia, A. Hulgeri, C. Nakhey, S. Chakrabarti, S. Sudarshan,"Keyword searching and browsing in databases using BANKS", In ICDE, 2002.

[3] S. Brin, L. Page,"The anatomy of a large-scale hypertextual web search engine", In WWW7, 1998.

[4] N. Bruno, L. Gravano, A. Marian,"Evaluating top-k queries over web-accessible databases", In ICDE, 2002.

[5] A. Burns,"Preemptive priority based scheduling: An appropriate engineering approach", In Advances in Real Time Systems, pp. 225–248. S. H. Son, Prentice Hall, 1994.

[6] R. Fagin, A. Lotem, M. Naor,"Optimal aggregation algorithms for middleware", In ACM PODS, 2001.

[7] D. Florescu, D. Kossmann, I. Manolescu,"Integrating keyword search into XML query processing", In WWW9, 2000.

[8] R. Goldman, N. Shivakumar, S. Venkatasubramanian, H. Garcia-Molina,"Proximity search in databases," In VLDB, 1998.

[9] L. Guo, F. Shao, C. Botev, J. Shanmugasundaram, "XRANK: Ranked keyword search over XML documents", In ACM SIGMOD 2003.

[10] V. Hristidis, N. Koudas, Y. Papakonstantinou,"PREFER: A system for the efficient execution of multi-parametric ranked queries", In ACM SIGMOD, 2001.

[11] V. Hristidis, Y. Papakonstantinou,"DISCOVER: Keyword search in relational databases", In VLDB, 2002.

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

Mr. K.Chandra Mouli received the Master of Computer Applications from Lakireddy bali reddy college of engineering, Mylavaram, Andhra Pradesh, India in 2007. He worked as lecturer in Computer Science. He is currently Pursuing M.Tech from Aditya Engineering College, Surampalem, Andhra Pradesh, India.

Mrs G.Aparna ,Working as Asst. Prof in the Department of Information Technology In Aditya Engineering College Surampalem. She Receved M.tech from Andhra University In Artificial Intelligence and Robotics. She Have 4 Years experience in various engineering Colleges. she designed and guided Many Projects.