DB2 Optimization in Support of Full Text Search

Albert Maier
IBM Boeblingen Lab
amaier@de.ibm.com

David Simmen
IBM Silicon Valley Lab
simmen@us.ibm.com

1 Introduction

Over the last several years, demand for integrating full text and relational search has increased dramatically. For example, a content management system might seek the most relevant articles written during a certain period, where the abstract contains the words 'Bush' in the same sentence as 'recession'. Neither a traditional text information retrieval (IR) system nor a traditional RDBMS could handle this type of query efficiently.

Not surprisingly, federated solutions emerged where an application layer decomposed queries into relational and text search components and joined the results. IBM's Digital Library exploited DB2 and IBM's powerful linguistic text search engine (TSE) in this way. One of the inherent drawbacks of this architecture is the cost of moving partial results into the application layer and joining them there.

DB2's Text Extender (TE) was the first attempt by a major RDBMS vendor to tackle this problem. TE integrates TSE into DB2. It supports creation of text indexes on DB2 tables. These text indexes use keys stored in the table to relate index entries back to individual records. Triggers are used to keep the text and text indexes in sync. TE exploited DB2's SQL extensibility features to effectively push query decomposition and join into the RDBMS. Following is an example of a typical TE query that performs the "recession" search described above:

```
SELECT a.isbn, a.year, b.score
FROM articles a, text_search('articles','abstract','"Bush" in same sentence as "recession"') b
WHERE a.year >= 2000 AND a.isbn = b.key AND b.score >= 0.9
ORDER BY 3
```

The table function *text_search* calls TSE to evaluate the text search condition, returning the keys of all matching articles together with a relevance score. This approach allows text and relational results to be joined using the sophisticated join methods of the RDBMS; however, optimal decomposition of the query, e.g., choice of access order, join methods, etc. requires information regarding the cost and cardinality of the text search be made available to the query optimizer. Note that performance of the query would also benefit if DB2 could make TSE aware that the query was interested in only the most relevant articles having a certain minimum score. So in addition to the cost estimation issues that effected performance, there were lost opportunities to limit cross-source data flow by pushing query processing (e.g., predicate application) to TSE.

These federated query optimization issues have been investigated extensively since TE was released. Of particular relevance here is the Garlic work [1], which described a general framework for performing cost-based

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

optimization of queries across heterogenous data sources. We exploited basic ideas from this work in order to solve many of the TE optimization issues described above. In particular, like Garlic, we did not wire knowledge about the capabilities of the text search engine into the DB2 optimizer. Instead, we added an API that allows the text search engine to exchange costing and planning information with the DB2 optimizer. In addition, we also added a set of built-in DB2 functions and related query rewrite support to allow for the most user-friendly formulation of SQL queries with full text search.

The remainder of this paper focuses on these features, which were added to DB2 Universal Database Version 7.2 (DB2 henceforth) in support of IBM's DB2 Text Information Extender (TIE) [2], the successor to TE. In addition to adding support for the exchange of information with the DB2 optimizer, TIE exploits IBM's next generation text search engine GT9, the successor to TSE. GT9 technology is also used in many other IBM products such as Lotus Notes, Enterprise Information Portal, and Intelligent Miner for Text.

2 DB2 Optimization in Support of Full Text Search

The DB2 SQL compiler is a direct descendent of Starburst [3] and generates a query execution plan similarly via parsing, semantics, query rewrite [4], and cost-based optimization phases. The following subsections describe changes made to the compiler to support full text search.

2.1 Semantics

DB2 provides 3 scalar functions in support of full text search. Each operates on a column and text search string.

- CONTAINS returns 1 if the text search string matches the first argument, and 0 otherwise.
- SCORE returns a float value between 0 and 1 indicating how well the text search condition was matched.
- NUMBEROFMATCHES returns an integer indicating how often it matched.

These functions serve only a semantic role. There is no supporting DB2 implementation. As will be discussed in the next subsection, a reference to these functions triggers a transformation to an equivalent query involving a table function supplied by TIE as a gateway to GT9.

This table function, which we call *text_search* for this exposition, takes as input meta-data identifying the text index that will support the text search, a text search string, a plan for evaluating additional parts of the query, and any input values required to evaluate the plan. The function returns the primary keys of all documents matching the text search condition as well as the columns *contains*, *score*, and *numberofmatches*, which supply the results for the respective DB2 scalar functions described above.

2.2 Query Rewrite

A scalar function interface provides the most user-friendly way to integrate text search, but a naive implementation could not exploit text indexes and would be very inefficient. The query rewrite component handles the burden of resolving TIE meta-data and composing joins with the *text_search* table function. To illustrate the basic idea, consider the following query that determines how relevant 2001 articles are with respect to the subject "G.W. Bush".

```
SELECT title, SCORE (a.abstract, '"G.W. Bush"' )
FROM articles a
WHERE a.year = 2001
```

Upon encountering the *score* function reference, DB2 retrieves TIE meta-data for the *abstract* column, resolves the primary key of the *articles* table, and transforms the query into the following:

SELECT a.title, COALESCE(b.score, 0)
FROM articles a LEFT JOIN text_search ('abstract_meta-data', '"G.W. Bush"', ...) b ON a.isbn = b.key
WHERE a.year = 2001

A left join operator is required to assure that 2001 articles not matching the text search condition are preserved. The coalesce function provides these rows with a relevance score of 0.

As illustrated with the following example, the rewrite logic assures that multiple scalar functions are mapped to the same table function output when possible:

SELECT title, SCORE (a.abstract, '"G.W. Bush"')
FROM articles a
WHERE a.year = 2001 AND CONTAINS (a.abstract, '"G.W. Bush"') = 1

The rewrite logic first processes the *score* function reference as in the previous example. It then encounters the reference to the *contains* function, and after comparing its corresponding index meta-data and text search strings with those of the existing table function, determines it can simply be mapped to the output of that function as follows:

SELECT a.title, COALESCE(b.score, 0)
FROM articles a LEFT JOIN text_search ('abstract_meta-data', '"G.W. Bush"', ...) b ON a.isbn = b.key
WHERE a.year = 2001 AND b.contains = 1

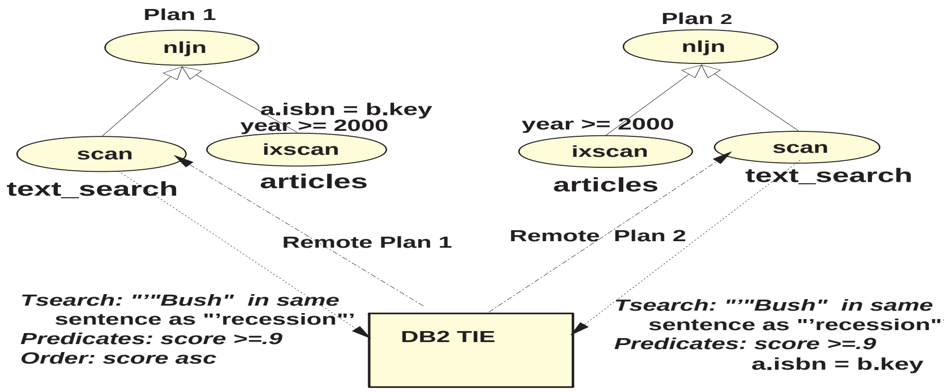
No COALESCE function is needed in this case since the optimizer's theorem-prover can determine that the conjunct *COALESCE(b.contains, 0)=1* rejects rows when instantiated with NULL. This is an important point, as it allows the outer join operator to be converted to an inner join operator later, by an existing rewrite rule.

2.3 Cost-based Optimization

Each time the DB2 optimizer evaluates an alternative plan for accessing the *text_search* table function, it calls the TIE optimizer to exchange costing and planning information. During each exchange, the DB2 optimizer presents the TIE optimizer with a set of *requirements* it would like TIE to satisfy. Examples of requirements includes the text search condition, predicates on output columns, output order, etc.

The TIE optimizer returns a *remote plan* for satisfying one or more of the requirements. The remote plan is treated as a black box by the DB2 optimizer. It is simply passed back to TIE when the *text_search* table function is opened for execution. A set of *properties* is associated with the plan. The properties describe which of the requirements the remote plan will satisfy (it will at least satisfy the text search condition), the cost of satisfying those requirements, and the cardinality of the result. The DB2 optimizer will add logic to the overall plan to compensate for any requirements not satisfied by TIE.

The figure below illustrates the interaction between the DB2 and the TIE optimizers. Two alternative evaluation plans for the query from Section 1 are shown. In plan 1, the *text_search* table function is accessed on the outer of a nested-loops join operation. In plan 2 the roles are reversed. The figure illustrates the sets of requirements passed to the TIE optimizer in each case. In both cases, the DB2 optimizer passes the text search condition and the predicate *score* $>=$ *.9* as requirements. The DB2 optimizer also includes the order requirement with the plan 1 requirements, since rows produced in score order would propagate through the join operation, satisfying the ORDER BY requirement. Remote plan 1 is passed back to TIE (argument 3) at execution time if plan 1 is cheapest overall. The DB2 optimizer forgoes adding the ordering requirement to the plan 2 requirements, but instead adds the join predicate *a.isbn = b.key*. If the join predicate is accepted by TIE, and plan 2 is cheapest overall, the *text_search* table function will be evaluated for each outer row. In this case, DB2 will pass outer row values needed to evaluate the join predicate (argument 4).



3 Related and Future Work

Integration of text search into an RDBMS is often done via some type of index extension mechanism [5], [6]. These approaches fail to exploit the complex query evaluation capabilities of advanced text search engines. For example, GT9 can evaluate predicates, produce ordered results, limit those results to only the first n rows, and so on. Modeling the search engine as a remote data source that participates in query optimization allowed us to exploit these capabilities. In the future, we would like to explore the idea of generalizing and exposing the costing and planning interface so that other text search engines can interact with DB2 in this way.

TIE is well-suited for content management applications that typically issue complex queries combining text and relational search conditions. These queries usually require complete answer sets. In contrast, e-commerce applications issue simpler queries with either no relational search conditions, or with only very simple ones. Moreover, these queries require only a few relevant results. IBM offers a specialized solution for e-commerce scenarios, called Net Search Extender (NSE). NSE exploits main memory techniques to achieve extremely high performance and scalability. TIE and NSE will be integrated into a single product in the future.

4 Conclusion

This paper describes DB2 optimization support for queries integrating relational and full text search, the key aspect of which is a costing and planning interface that allows our text search engine to fully participate in determination of the optimal query execution plan. The paper also describes how rewriting improves usability and performance. Special thanks to Laura Haas for reviewing an earlier draft of the paper.

References

[1] L. Haas, D. Kossmann, E. Wimmers, and J. Yang: *Optimizing Queries Across Diverse Data Sources*, VLDB'97

[2] Text Information Extender: *http://www.ibm.com/software/data/db2/extenders/textinformation*

[3] L. Haas, J. Freytag, G. Lohman, and H. Pirahesh: *Extensible Query Processing in Starburst*, SIGMOD'89

[4] H. Pirahesh, J. Hellerstein, W. Hasan: *Extensible Rule-based Query Rewrite Optimization in Starburst*, SIGMOD'92

[5] J. Srinivasan et al.: *Extensible Indexing: A Framework for Integrating Domain-Specific Indexing Schemes into Oracle8i*, ICDE 2000

[6] Informix Dynamic Server V9.3, *Virtual Index Interface, Programmers Manual*, Part No. 000-8345, August 2001