Text / Relational Database Management Systems: Harmonizing SQL and SGML

G. E. Blake, M. P. Consens, P. Kilpeläinen, P.-Å. Larson, T. Snider, and F. W. Tompa

UW Centre for the New OED and Text Research,
University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

Abstract. Combined text and relational database support is increasingly recognized as an emerging need of industry, spanning applications requiring text fields as parts of their data (e.g., for customer support) to those augmenting primary text resources by conventional relational data (e.g., for publication control). In this paper, we propose extensions to SQL that provide flexible and efficient access to structured text described by SGML. We also propose an architecture to support a text/relational database management system as a federated database environment, where component databases are accessed via "agents": SQL agents that translate standard or extended SQL queries into vendor-specific dialects, and text agents that process text sub-queries on full-text search engines.

1 Introduction

The application of database technology is seen as essential to the operation of a conventional business enterprise. However, there is a universe of business information, namely text, which is currently stored, accessed, and manipulated in an *ad hoc* fashion with none of the consistency and discipline of the database approach. Environments supporting both text and relational data are implemented through application programs within which separate repositories are accessed explicitly. Not only is this inconvenient for application programmers, but the disjointness of the data impedes data administrators' efforts to ensure data consistency. Furthermore, the difficult task of query optimization becomes the burden of every application programmer and the benefits of database transparency are impossible to realize. Ongoing work at the University of Waterloo has laid the foundations necessary for building an alternative to this disorder and lost potential.

The objective of the research is to design and implement a multidatabase system supporting text and relational data (T/RDBMS) that will better address the needs of these enterprises. We start with the requirement that the application

Published in Proceedings of the ADB'94 Conference.

program interface must be an extension of both SQL, the industry standard for relational data [ISO90], and SGML, the industry standard for structured text [Gol90, ISO86].

The T/RDBMS can be built as a federated database system with the actual data stored and managed by standard (relational and text) data management systems. Queries expressed in terms of the external data model are parsed, and the relational and text components identified. Query strategies can then be analyzed so that an efficient access plan can be identified. This plan can subsequently be executed under the control of a database monitor, which distributes parts of the query task to component database systems as needed, and integrates the results before they are returned to the application.

Several approaches to text management have been proposed by others. Customized document storage management systems, including text-specific access languages, have been implemented on top of commercial relational database systems (see, for example, [Wei85, Mar91]); these systems are incapable of simultaneously supporting conventional data. Alternatively, text storage has been provided by conventional systems, where long data fields are used for large objects or "blobs" [Bil92], but operators to support text manipulation have not usually been provided and these systems do not support SGML-like structured text.

At least three systems have been proposed within which structured text can be fragmented into relational fields and SQL queries can be applied against the resulting text subfields in conjunction with record-oriented data. The Air Transport Association has proposed the Structured Full-text Query Language (SFQL) as an extension to SQL incorporating SGML-based formatted text types [ATA91]. More recently, Oracle Corporation's SQL*TextRetrieval Version 2 provides a text retrieval product, supported by inverted indexing and a thesaurus capability, to be used in conjunction with the Oracle DBMS [Ora92]. Similarly, IDI's BASISplus supports structured full-text retrieval in conjunction with relational database functionality [Sey92]. Although each system provides a mechanism to assemble larger text units from the constituents, this is not provided within SQL. Thus, for example, such larger units cannot be presented as fields within an SQL view.

In order to maintain structured text in a single relational field, researchers at Australia's Collaborative Information Technology Research Institute have designed and implemented a nested relational database system (Atlas) and an extended SQL language to provide text support [Sac92]. Similarly there is a recent proposal to extend an object-oriented SQL dialect to support SGML documents [Chr94]. In both these approaches, the relational model has been extended to encompass structured data of arbitrary type, and subsequently structured text has been supported as a special case. We instead wish to explore a direct extension of SQL to support structured text in the hope that our proposals will suit text, and particularly SGML applications, more closely.

2 Example Text Database

To illustrate our proposed text extensions to SQL, we will use a simplified version of a database management system required for an encyclopedia, and describe our extensions in terms of this example.

Such a database requires management of both administrative records and text. Information about contributors and their articles, including tracking the development of the articles, must be maintained. In addition text management involves key-word generation, cross-reference maintenance, maintaining consistency of style, and maintaining consistency of bibliographic data.

Standard SQL type queries against this database are to be supported, to extract contributor addresses in order to generate address labels or to extract information about articles having due dates in a given time range, and to check information about payments to contributors. In addition, queries are to be expected against the bibliographic data using a variety of criteria based on authors, dates, and number of citations. Similarly, queries posed against the content of articles must be supported.

The articles themselves contain primarily prose text, but consider the form of the bibliography. Bibliographic data can be presented in a list format, a prose format or a combination of these two. For example, the bibliography for the article entitled "Canada, History of" in *The New Encyclopæaedia Britannica — Macropaedia* Vol. 3, p. 751, reads:

> W.L. MORTON, *The Kingdom of Canada*, 2nd ed. (1969), is the fullest one-volume history and the most traditional.... To understand the place of the colonies that became Canada in the British Empire, the following are most useful: H.A. INNIS, *The Fur Trade in Canada*, 2nd ed. (1956), and *The Code Fisheries*, rev. ed. (1954);... The following works both introduce and analyze the development of the remaining British colonies to self-governing communities and their union in confederation. W.S. MACNUTT combines in a single narrative the histories of the Atlantic provinces in *The Atlantic Provinces, the Emergence of Colonial Society, 1712-1857*(1965). FERNAND OUELLET in his *Histoire 'economique et sociale du Qu'ebec, 1760-1850* (1966; Eng. trans. in prep.), applies with great success the demographic method of French historiography to the little known domestic development of that province....

A requirement for this database is that we must be able to retrieve the articles and bibliographies in their text form as crafted by the encyclopedia's contributors and editors. Thus, for example, we must be able to deal with the bibliography *as a single structured textual unit* and yet extract individual authors or citations as structured texts.

To illustrate our proposed language extensions, we will define one simplistic table with the following schema.
where *aid* is the article identification, *title* is the proposed article title, *cid* is the contributor identification, *req_date* is the date the article was solicited from the author, *req_wc* is the requested word count for the article, *due_date* is the requested date for the article's completion, *article* is the text of the completed

Encyclopedia

aid	title	cid	req_date	req_wc	due_date	article	biblio

article, and *biblio* is the accompanying bibliography. For the sake of brevity and clarity, we will describe constructs in the DDL and DML in terms of this example; formal definitions and more detailed explanations can be found elsewhere [Bla94].

3 Data Definition Language

In our example the *article* and *biblio* fields are of a new data type TEXT. Fields of this type contain structured text and have an associated grammar to describe their content. Queries involving this new data type may use this grammar. The grammar for any of these TEXT fields must therefore be made known to the database.

The following SGML Document Type Declarations (DTDs) describe the grammar of *article* and *biblio* TEXT fields in our example. Note that the article may consist of a cross reference to another article or may itself be a complete article. The body of the article is followed by some keywords and some summary information that contains data such as birth place and dates of the article's subject, as appropriate. The bibliography allows free text to be intermingled with bibliographic fields as desired.

```
<! DOCTYPE article_information [
    <! ELEMENT article_information - - (detailed_article|xref)>
    <! ELEMENT xref - - ("see", article_title)>
    <! ELEMENT detailed_article - - (title,paragraph+,keywords*,summary)>
    <! ELEMENT summary - - (birth|death|parents|occupation|children)*>
    <! ELEMENT (title,keywords,paragraph,article_title) - - PCDATA>
    <! ELEMENT (birth,death,parents,occupation,children) - - PCDATA>
]>
```

```
<! DOCTYPE biblio_information [
    <! ELEMENT biblio_information - - (citations)>
    <! ELEMENT citations - - (citation, (";" | "."))+>
    <! ELEMENT citation - - (author|ref|date|free_text)+>
    <! ELEMENT ref - - ( "in"?,work,edition?) >
    <! ELEMENT date - - ( "(", PCDATA , ")" )>
    <! ELEMENT (author,work,edition) - - PCDATA>
    <! ELEMENT free_text O O PCDATA>
]>
```

To make known the grammar for any defined TEXT fields, we use a CREATE GRAMMAR statement, which incorporates standard SGML notation for this purpose.

```
CREATE GRAMMAR (<! ENTITY %example SYSTEM "/usr/dtd/art_inf"> %example;);
```

where the article_information DTD is assumed to be stored in the file named "/usr/dtd/art_inf". Such a statement informs the system where an SGML document entity is located. The document entity contains the SGML declaration (if necessary) and a prolog containing a DTD. A similar statement is required to introduce the grammar for the biblio_information DTD.

Now we can define the database table in the following CREATE TABLE statement.

```
CREATE TABLE Encyclopedia (
              aid INTEGER, title STRING, cid INTEGER,
              req_date DATE, req_wc INTEGER, due_date DATE,
              article TEXT GRAMMAR article_information,
              biblio TEXT GRAMMAR biblio_information
              PRIMARY KEY (aid)
              );
```

This statement defines a table called *Encyclopedia* with eight columns. Two are of the new type TEXT, for which we must give the name of an associated grammar.

In this case the primary key is an INTEGER column. However, the primary key for a table may instead be declared to be a subfield described by a nonterminal in the grammar associated with one of the TEXT columns.

We may want to extract certain subfields of TEXT columns in order to define a view. Consider, for example,

```
CREATE VIEW cited_auths AS
   (SELECT aid, title, biblio..author FROM Encyclopedia);
```

which defines a three column table containing the article's identification, title and cited authors. The semantics of elements such as biblio..author appearing in a SELECT are given in the next section.

In summary, the CREATE TABLE statement has been extended to accept a new data type for a column, namely TEXT. Unlike previous proposals, the type TEXT refers to *structured* searchable text, with an associated grammar. To comply with emerging text standards, we assume the grammar to be an SGML grammar, either one described by a DTD or derived as a sub-grammar of a given DTD, rooted at one of its elements. Elements of the grammar will also be available to be used in a query to refine a text search or to recover information about the grammar itself. Finally, primary and foreign keys may refer to data within a nonterminal in the grammar associated with a TEXT column.

4 Data Manipulation Language

In our attempts to combine the concepts of text and relational databases we have taken the approach that the text is embedded in relations rather than the other way around. Thus in our DDL we allow a field in a relation to be of type TEXT (i.e., structured text), with an associated grammar. In the DML we continue on this course: operations are typically applied to one structured field at a time.

For the purposes of the DML, an instance of a structured text field is considered to be a contiguous text string and a parse tree. Previous authors have proposed extensions to traditional SQL operators to include operations on *unstructured* text fields, in particular concatenation and pattern matching [ATA91, Ora92, Sac92]. For now we will accept those extensions and propose an operator and alternative notation for *structured* text manipulation suitable for extending SQL.

4.1 Examples of the Proposed DML

Before we explain the details of our proposed DML, we give two examples of its use. For convenience, we introduce a notation for projecting text subfields from attributes of type TEXT. In the next section we introduce a mechanism for selecting nodes in a parse tree, and we introduce a new operator, EXPAND, to extract some data from the paths to those nodes into a relation. Once we have described EXPAND, the semantics of this first notation will be formally specified, after which we will return to these sample queries.

In our example database we might want to answer the question

> *Who contributed articles for which the proposed titles do not match the titles included in the article's body?*

With our extended SQL, we can formulate this query as

```
SELECT cid, aid
FROM Encyclopedia
WHERE title ≠ article..title;
```

Within the TEXT attribute named "article", title is a text subfield identified by an element of the associated SGML grammar. The dot notation gives direct access to such fields, and is available anywhere within the SELECT statement.

As a second question, consider

> *Find proposed titles and lengths of long articles on Canada.*

Using our DML the answer can be obtained with the query

```
SELECT title, req_wc
FROM Encyclopedia
WHERE req_wc > 5000 AND SOME article..keywords CONTAINS "Canada";
```

Notice that an article having more than one keyword is selected if any one matches the given search string.

4.2 Converting Parse Trees to Relations

For some queries we wish to convert parts of a TEXT column into a relation so that we have the full power of SQL to manipulate subfields. An instance of a structured text field is a contiguous text string together with a parse tree [Gon87]. In order to minimize the size of the relation formed from this parse tree we wish to select nodes in the parse tree to indicate the components of the text that we wish to retain.

Consider the first example above. We wish to compare the value in one attribute against the value stored within a subfield of another attribute. To extract the title element from the article column, we can use the EXPAND operator as follows:

```
SELECT cid, aid
FROM Encyclopedia
WHERE title ≠ (EXPAND article INTO title AS contrib_title BY
                  (SELECT nodeid FROM TEXT_NODES WHERE genid = "title"));
```

The EXPAND statement processes a parse tree and returns a relation. In this example, because one field is named between INTO and BY, the resulting relation will have a single column, populated by string(s) subsumed by node(s) of type title but renaming the single column as contrib_title.

Since in general we might not want to extract every node of a given type, the body of EXPAND consists of a SELECT statement that identifies desired nodes in the parse tree. So as to allow node selections to be written in full SQL, we define three *virtual* tables as follows:

TEXT_NODES (nodeid, genid, content)
TEXT_ATTRIBUTES (nodeid, attr, value)
TEXT_STRUCTURE (a_nodeid, d_nodeid)

where TEXT_NODES contains one tuple per node in the parse tree, consisting of a unique nodeid, the type of node ("generic identifier" in SGML), and the TEXT content subsumed by that node; TEXT_ATTRIBUTES relates SGML's attribute-value pairs to corresponding nodes; and TEXT_STRUCTURE contains nodeid pairs representing all ancestor-descendant relationships in the parse tree. Because these tables merely provide a notation to access data of type TEXT, the values for nodeid are undefined outside the EXPAND statement; within the EXPAND statement, however, nodeids can be compared for equality or for relative magnitude, where $n_1 < n_2$ if n_1 occurs first in a preorder traversal of the parse tree.

We illustrate the use of these tables with several examples, each returning a set of nodeids for selected nodes from the parse tree:

- *Select all nodes of type "paragraph" and containing substring "Canada".*

```
SELECT nodeid
FROM TEXT_NODES
WHERE genid = "paragraph" AND content CONTAINS "Canada"
```

Pattern matching uses the clause CONTAINS from previous SQL extensions.

- *Select all nodes having an ancestor of type "chapter".*

```
SELECT d_nodeid
FROM TEXT_STRUCTURE, TEXT_NODES
WHERE a_nodeid = nodeid AND genid = "chapter"
```

- *Select all nodes that have children.* (i.e., mark all *structured* fragments.)

```
SELECT DISTINCT a_nodeid
FROM TEXT_STRUCTURE
```

- *Select all nodes where the parent has an (SGML) attribute of type status and value "Obs."*

```
SELECT d_nodeid
FROM TEXT_STRUCTURE
GROUP BY d_nodeid
HAVING max(a_nodeid) IN SELECT nodeid
                        FROM TEXT_ATTRIBUTES
                        WHERE attr = "status" AND value = "Obs."
```

Having selected desired nodes in a parse tree, users might wish to extract any values along the paths from the root to those nodes. For example, having identified nodes corresponding to certain cited authors, a user might wish to extract complete citations as well as the authors' names. A corresponding subquery would be

```
EXPAND biblio INTO citation, author BY SELECT ...
```

where the SELECT clause identifies nodeids for nodes corresponding to relevant authors. If the SGML grammar defines the attribute "status" for elements with generic identifier "citation", the value of the attribute can also be extracted into a separate column:

```
EXPAND biblio INTO citation [status], author BY SELECT ...
```

In general, the EXPAND operator takes the form

```
EXPAND <col_reference>
INTO ([<node_name> [ <attr_list> ] AS] <col_name>)+
BY <select_stmt>
```

where attr_list is `"[" ([<attr_name> AS] <col_name>)+ "]"` . Thus several nodes' contents and their attribute values can be extracted, and each can be renamed.

To populate the resulting relation, we trace the paths from the root to each selected node in the parse tree and form one tuple for each such path (i.e., one tuple per selected node). Specifically, for each selected path, nodes on the path are examined in order from the root to the leaf and the node name list on the EXPAND line is scanned from left to right: as a node of the same type as specified in the list is found, its *content* (of type TEXT) is transferred to the associated column in the row and any designated attribute *value* is extracted as well. Any fields in the generated row for which there is no match along the path is given a NULL value.

4.3 Examples of SQL Queries Using EXPAND

For our example we continue to use the sample bibliographic database.

Find the article identification, title, contributor, and bibliography for entries where the bibliography has more than one citation but all are from the same author.

```
SELECT aid, title, cid, biblio
FROM Encyclopedia
WHERE 1 <SELECT COUNT(citation)
        FROM (EXPAND biblio INTO citation BY
              SELECT nodeid FROM TEXT_NODES WHERE genid = "citation")
AND 1 = SELECT COUNT(DISTINCT author)
        FROM (EXPAND biblio INTO author BY
              SELECT nodeid FROM TEXT_NODES WHERE genid = "author");
```

The EXPAND statement forms the basis of the dot operator used to simplify access to text subfields as in the examples shown earlier. For the expression $F..a_1..a_2 \ldots a_{n-1}..a_n$ describing a path in the parse tree of an instance of a TEXT field F in some table, we give the semantics:

```
EXPAND F INTO a_n BY

SELECT nodeid FROM TEXT_NODES, TEXT_STRUCTURE
WHERE nodeid = d_nodeid AND genid = "a_n"
AND a_nodeid IN (

            SELECT nodeid FROM TEXT_NODES, TEXT_STRUCTURE
            WHERE nodeid = d_nodeid AND genid = "a_{n-1}"
            AND a_nodeid IN (

            ...

                        SELECT nodeid FROM TEXT_NODES, TEXT_STRUCTURE
                        WHERE nodeid = d_nodeid AND genid = "a_1"

            ) ...)
```

Extraction of attribute value b from the selected node can be simply specified as $F..a_1..a_2 \ldots a_{n-1}..a_n[b]$.

Consider again the second example from Section 4.1:

```
SELECT title, req_wc
FROM Encyclopedia
WHERE req_wc > 5000 AND SOME article..keywords CONTAINS "Canada";
```

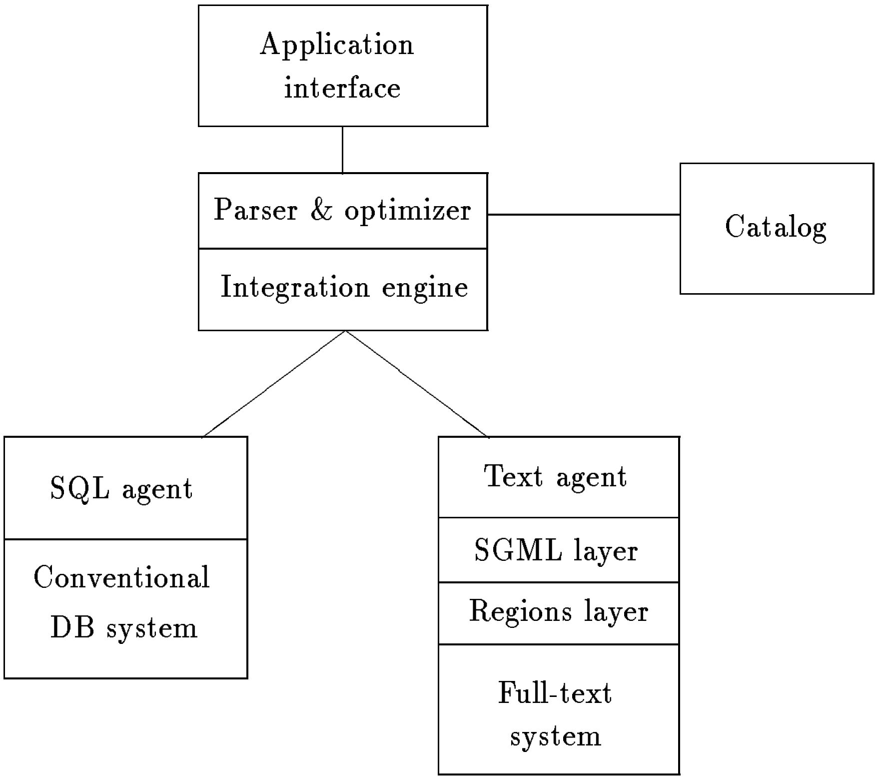
This is now well-defined as:

```
SELECT title, req_wc
FROM Encyclopedia
WHERE req_wc > 5000
AND SOME (EXPAND article INTO keywords BY
          SELECT nodeid FROM TEXT_NODES WHERE genid = "keywords")
          CONTAINS "Canada";
```

Complete semantics for uses of the dot notation throughout the SELECT statement (including in the SELECT and FROM clauses) have also been defined [Bla94].

5 Federated Database Architecture

Potential clients of the text/relational database management system have typically relied on conventional relational databases for conventional needs. Therefore we propose not to develop a wholly new database management system, but instead to integrate commercial relational database systems and commercial full-text search engines. To this end we define the following architecture:



Application interface: This component provides an interface through which an application program interacts with the T/RDBMS system. It consists of a set of functions for submitting requests and retrieving the result of a request. In support of a data model integrating text with conventional relational data, requests are expressed in a version of SQL extended with features for text searching and text manipulation.

Parser & optimizer: When a request arrives from an application program, it is first parsed and semantically analysed. The next step is query optimization, that is, determining an efficient way of servicing the request, which is crucial to achieving good performance. These two steps require access to text and relational data descriptions, which are stored in a catalog (also called data dictionary). The output of these steps is a self-contained access plan that specifies how to evaluate the request: what requests (queries) to submit to each component system and what operations to perform to integrate the data from the various sources.

Integration engine: The access plan is passed to the integration engine which performs the computation specified in the plan. This consists of retrieving data from the underlying data sources (i.e., relational database system and text management system) by submitting requests specified in the plan and performing the processing required to integrate data from the several sources. Integration typically requires joins, but other operations may also be involved. The result is then returned to the application.

SQL agent: The system should not be dependent on any particular database system. The purpose of an SQL agent is to hide the specifics of the underlying database system and present a standard interface. We have already developed such agents for several database engines [Lar89a, Lar89b] and expect to develop additional ones in the near future. Each agent has a call-level interface based on Microsoft's Open Database Connectivity (ODBC) specification [Mic92].

Text agent: Similarly, the system should not be dependent on any particular text management system. Text agents providing a common interface to different text systems are being developed. However, no widely accepted, common interface exists in this area. We are currently implementing two layers between the central component and flat-text engines, one that converts our grammar-based queries into a form based on simpler "regions" or "zones" (such as those supported by PAT [Sal92]), and a second one that eliminates dependence on any structural capabilities being provided by the text search system. With this three-layer approach to text agents, the system will be compatible with text systems having varying degrees of obliviousness to structure.

The proposed architecture is based on our experience with multidatabase systems that integrate more traditional database system. Its main design objectives are flexibility and independence of underlying data sources (conventional database systems and text systems). Independence is achieved by providing agents that present a standardized interface and hide the details of of the underlying system. There is a price to be paid for this: the translation performed by an agent requires some extra processing, and, more significantly, the system may not be able to exploit some of the features of an underlying database or text system. However, the overall system is greatly simplified by isolating the rest of the software from the details of those systems. Not only is this good software practice, but it allows the integration of commercial database management systems into the architecture, where those systems can retain their independence and integrity. Furthermore, the architecture supports distribution: various components may be implemented as separate processes, possibly running on different machines.

Given performance estimates for each client subsystem, the central database engine must choose in which order and by which means to process the user's request. Consider again the simple query for long articles on Canada. If the requested length is stored in a traditional relational database (as a field in Encyclopedia_relation) and keywords are stored in a text database (in Encyclopedia_text), this could be implemented by

```
SELECT title, req_wc
FROM Encyclopedia_relation
WHERE req_wc >5000 AND aid IN (SELECT aid
                               FROM Encyclopedia_text
                               WHERE article..keywords CONTAINS "Canada");
```

Is it more efficient to find the relational records for long articles and then retrieve corresponding text records that include "Canada" among the keywords; or to find all text records that include the keyword "Canada", extract the article identifier and use that as part of a conventional relational subquery; or to execute both subqueries in parallel and then join the results? At least superficially, these choices correspond to those faced by global query optimization algorithms for conventional distributed and federated database systems, but we have learned from past experience that text brings its own needs and surprises to traditional database problems.

6 Conclusions and Further Work

We have developed a single model within which both text and relational data can be described so that users can access and manipulate all their data meaningfully. Our proposed extensions to SQL are modest, yet they are powerful enough to handle SGML-based data simply, to support extractions from highly structured text into relations, and to preserve the integrity of complex text units. We have designed a data description language and a query language integrating SQL with text search and text manipulation features, addressing the following questions:

- How can an SGML-defined description of text be integrated with SQL's DDL?

- Which text manipulation operators should be included in an extended SQL DML?

We have not yet designed update extensions to SQL that permit text subfields to be modified.

We have designed an architecture to support integrated text-and-relational databases using a federated database system. We have begun to implement and test the architecture and language, supporting data stored partially in an Oracle relational database and partially under the control of the PAT text engine. We will test all three text interfaces (SGML, regions, and flat-text) against PAT, and we expect that our experience will be transferable to other engines with various capabilities. We do not expect full-text search engines to have sufficiently fast update characteristics to support transaction rates common to relational database environment, and we plan to investigate the development of faster and more flexible text update algorithms.

In further research, a primary problem to be addressed is execution planning. Before access strategies can be selected on the basis of expected performance, estimates of costs must be made available to the query optimizer. To achieve this end, traditional approaches require that we first develop cost models for text processing systems. In particular, for each class of operations, we must be able to determine upper and lower bounds on access time as well as expected access time to complete an operation in that class. We must then devise query optimization and data integration strategies that take advantage of such information.

References

[ATA91] ATA 89-9C SFQL Committee, "Advanced Retrieval Standard —SFQL: Structured Full-text Query Language," *ATA specification 100, Rev 30, Version 2.2, Prerelease C,* Air Transport Association, ATA 89-9C.SFQL V2.2/PR-C (October 1991) 84 pp.

[Bil92] A. Biliris, "The Performance of Three Database Storage Structures for Managing Large Objects," *Proc. Sigmod 92,* ACM, *Sigmod Record*, Vol. 21, No. 2 (June 1992) 276–285.

[Bla94] G.E. Blake, M.P. Consens, P. Kilpeläinen, P.-Å. Larson, T. Snider, and F.W. Tompa, "Text extensions to SQL," internal report, Univ. of Waterloo Centre for the New OED and Text Research, 1994.

[Chr94] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl, "From Structured Documents to Novel Query Facilities," To appear in *Proc. 13th. ACM SIGMOD Conf.*, (May 1994).

[Gol90] C. F. Goldfarb. *The SGML Handbook.* Oxford University Press, Oxford, 1990.

[Gon87] G. H. Gonnet and F. W. Tompa, "Mind Your Grammar: a New Approach to Modelling Text," *Very Large Data Bases* (VLDB), Vol. 13 (September 1987) pp. 339–346.

[ISO86] International Organization for Standardization, *International Standard 8879: Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML)*, first edition — 1986-10-15(Ref. No. ISO 8879-1986(E)), 155 pp.

[ISO90] International Organization for Standardization, "Information technology – Database Language SQL 2 Draft Report", ISO Committee ISO/IEC JTC 1/SC 21, 1990.

[Lar89a] P.-Å. Larson, "Relational Access to IMS Databases: Gateway Structure and Join Processing," Project report, (available from the author), 1989.

[Lar89b] P.-Å. Larson, H. AboElFotoh, M. Dionne, and F. Wang, "SQL Access to VAX DBMS Databases: Strategy Generation and Query Execution for Basic SFW Queries," Project report (available from the first author), 1989.

[Mar91] C. C. Marshall, F. G. Halasz, R. A. Rogers, and W. C. Janssen Jr., "Aquanet: a Hypertext Tool to Hold Your Knowledge in Place," *Proc. 3rd ACM Conf. on Hypertext: Hypertext 91*, San Antonio (Dec. 1991) 261–275.

[Mic92] Microsoft Corporation, *Microsoft ODBC Application Programmer's Guide*, Microsoft Corporation, 1992.

[Ora92] Oracle Corporation, *SQL*TextRetrieval Version 2 Technical Overview*, Oracle Corporation, 1992. 45 pp.

[Sac92] R. Sacks-Davis, A. Kent, K. Ramamohanarao, J. Thom, , and J. Zobel, "Atlas: a nested relational database system for text applications", Technical Report CITRI/TR-92-52, Collaborative Information Technology Research, Victoria, Australia, July 1992.

[Sal92] A. Salminen and F.W. Tompa. "PAT Expressions: an algebra for text search," *Papers in Computational Lexicography: COMPLEX '92*, Proc. 2nd Int. Conf. on Computational Lexicography (F. Kiefer, G. Kiss, J. Pajzs ed.), Linguistics Inst., Hungarian Academy of Science, Budapest (October 1992), 309–332.

[Sey92] Seybold Publications, "IDI Pursues Document Management," *Report on Publishing Systems*, Vol. 21, No. 16, May 1992.

[Wei85] E.S.C. Weiner. "The *New OED*: Problems in the Computerization of a Dictionary," *University Computing*, Vol. 7 (1985) 66-71.

[Zlo75] M.M. Zloof. "Query-by-Example: Operations on the Transitive Closure," IBM Research Report RC 5526, Yorktown Heights, N.Y., 1975.

Acknowledgements

This work has benefited from discussions with Tim Bray, Gordon Cormack, and Gaston Gonnet. The financial assistance of the University of Waterloo, the Natural Sciences and Engineering Research Council of Canada, and Open Text Corporation are gratefully acknowledged.

This article was processed using the LaTeX macro package with LLNCS style