A Structured Text ADT for Object-Relational Databases

L.J. Brown, M.P. Consens, I.J. Davis, C.R. Palmer, and F.W. Tompa

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

July, 1997

ABSTRACT

There is a growing need, both for use within corporate intranets and within the rapidly evolving World Wide Web, to develop tools that are able to retrieve relevant textual information rapidly, to present textual information in a meaningful way, and to integrate textual information with related data retrieved from other sources.

This paper introduces a model for structured text and presents a small set of operations that may be applied against this model. Using these operations structured text may be selected, marked, fragmented, and transformed into relations for use in relational and object oriented database systems.

The extended functionality has been accepted for inclusion within the SQL/MM standard, and a prototype database engine that supports SQL with extensions to incorporate the proposed text operations has been implemented. This prototype serves as a proof of concept intended to address industrial concerns, and it demonstrates the power of the proposed abstract data type for structured text.

1. Introduction

The application of database technology is essential to the operation of conventional business enterprise, and it is becoming increasingly important in the development of distributed information systems. However, most database technology, and in particular relational database technology, provides few facilities for effectively managing the vast body of electronic information embedded within text.

Recognizing that the potential exists for text retrieval to become of much greater relevance within information systems, Fulcrum Technologies Inc., Grafnetix Systems Inc., InContext Corporation, Megalith Technologies Inc., Open Text Corporation, Public Sector Systems, and SoftQuad Inc., and the University of Waterloo formed the Canadian Strategic Software Consortium (CSSC) in 1993, in order to pursue pre-competitive research relating to the integration of relational databases and text-intensive databases [CSSC94]. Working with other members of CSSC, the University of Waterloo decided to explore how relational database systems could be extended, so that they might most effectively provide access to structured text in a manner compatible with SQL [Bla94, Bla95]. Past experience suggested that large texts need to be capable of being searched both vertically with respect to their internal structure and horizontally with respect to their textual content [Wei85]. They need to be capable of being fragmented at appropriate structural boundaries, so that the appropriate context surrounding the selected text can be recovered, and selected text needs to be marked in some manner, so that it can subsequently be located easily within a potentially much larger context [ATA91].

Early versions of our SQL2 extensions have been presented with examples [Bla94, Bla95]. The semantics of these extensions have since been refined and formally defined [Dav96], and the extensions have subsequently been adopted for inclusion within the SQL/MM standard [ISO96s].

We were at the time also actively researching how federated database systems might be constructed on top of existing database and text searching systems, such as Oracle, IMS, DB2/6000, and PAT [Cob92, Zhu92]. We therefore elected to build a prototype hybrid query processor capable of integrating relational data (managed by relational database systems) and text (managed by text engines) [Bri97].

In this paper we describe the final text model that we developed to meet the varied uses of structured text in a database environment. Section 2 highlights the diverse nature of text. The structured text abstract data type is presented in Section 3. Section 4 describes a set of related applications that illustrate the utility of such structured text objects, and conclusions follow in Section 5.

2. The challenge

A structured text is by definition any text that has an identifiable internal structure. This structure may be explicitly established by the inclusion of appropriate electronic markup [Coo87, Tom89], possibly complemented by an external document type definition (DTD), or it may be implied by the language contained within this text. HTML is an example of a text that contains explicit structural markup used in association with a DTD [Rag97], while Java source code is an example of text whose structure is determined by appropriately parsing the language contained within the text [Fla96].

In general, structures within a text may be arbitrarily complex. Regions of structured text (i.e., instances of subtext spanning a well-defined textual region) may overlap, and referential relationships between text may form complex interwoven networks both within texts and across texts [Spe94]. These networks may themselves not be particularly well defined, since in some cases a reference will be translated into a request to include subtext (*cf.* macro expansions), while in others it will establish a cross-reference to a separate subtext (*cf.* function invocations).

Structured subtexts may contain or reference heterogeneous objects such as image, sound, video, spatial and other data types frequently found within existing multi-media documents, and these objects may themselves contain or reference further instances of structured text. Structured texts may also exist as temporal objects, having differing manifestations at different times, all of which must be accessible by specifying the appropriate temporal context.

The ordering of text may be only partially defined, both within specific regions of structured text (e.g., SGML attribute values are unordered) and within the networks that arise when subtexts reference other subtexts. Certain text may be considered to be logically present while being physically absent (e.g., SGML attributes' values and C++ function parameters may be assigned default values when absent), while other text may be physically present but considered within some contexts to be logically absent (e.g., color, white space, punctuation, descriptive markup, versioning).

Text extracted from its original context poses even more problems. It is difficult to associate an appropriate grammatical structure with instances of extracted subtext when the context from which these text has been extracted is lost. For example, if three paragraphs are extracted from a chapter that forbids the inclusion of footnotes, do the extracted paragraphs continue to exclude footnotes? Having been extracted, what textual (or other) structure now contains the three extracted paragraphs? Does this containing structure allow other than three paragraphs within it, and does it associate any ordering with the paragraphs it contains? If a page break occurs between two of these three paragraphs in the original source, is this page break also present within the resulting structure associated with the three extracted paragraphs?

Developing a text model capable of representing a vast variety of properties that any possible application could demand of structured text would probably not be particularly constructive, even if it were possible. What is initially required is a simple text model that encapsulates most of the significant properties of structured text, coupled with well-defined operations on this model that facilitate effective query, retrieval and update of selected fragments of structured text [Ray96a].

3. Structured text objects

In this section, we describe a tree model for structured text, and we describe operations on that model that provide the functionality needed to select and extract subtexts in a relational database environment. The text operations have been implemented using native language constructs supported by Open Text's PAT 5.0 text engine [OTC95] and the University of Waterloo's MultiText text engine [Cla94]. Using ODBC [Mic92], output from these operations were integrated with texts stored in a Fulcrum database [Ful94] and with relational data stored in Oracle and DB2 databases. Similar implementations could be written to interact appropriately with object-oriented database systems and with other structured text search engines, using alternative native search languages.

3.1 A tree model for structured text

A structured text subsumes a region of text which may itself contain well-formed subordinate structured texts, such as a chapter containing paragraphs, footnotes, figures, and subchapters. It is assumed that a structured text is finite and that an arbitrary ordering of text may be associated with unordered fragments of text. For example, SGML attributes are considered to be logically unordered, and subtexts drawn from a collection of works contained within a single text may have no logical ordering; nevertheless the text is arbitrarily ordered in its presentation. Using these assumptions a structured text can be conceptually represented as an ordered tree having nodes that correspond to the various structures in the text [Mac92, Sal96]. Each node in this tree is labelled with a string that identifies the function of the structure that the node conceptually represents, and each node contains as a second attribute the subtext subsumed by this structure.

No assumptions are made in the model about what constitutes structural information within an arbitrary text; this must be decided by the process that parses a character string to interpret it as a structured text. Similarly, no assumptions are made about how the physical structure within the text is encoded within the model, since such assumptions would limit the usefulness of the model [Mac92].

This framework allows the full generality of the structured text model to be exploited by diverse applications. Application designers determine the structures that are to be identified in the model and how type information is to be encoded within node labels. Data providers choose the mechanisms for encoding structured text as character strings and ensure the existence of parsers that can be used to interpret strings' values as structured text, and thus populate the model. Through the use of standards such as SGML [ISO86] and DSSSL [ISO96d], applications and data providers can ensure that the data stored within the model exhibits the appropriate conceptual structures within a text. Thus the data provider assumes responsibility for the management of physical texts, the model provides a mechanism for describing how these physical texts may be accessed, and the data consumer remains responsible for deciding how a text is to be interpreted and manipulated.

As an example, consider the fragment of the University of Waterloo calendar [UW96] shown in Figure 1. This fragment could be encoded as tagged text as in Figure 2(a) and interpreted by a parser as the labelled tree shown in Figure 2(b),where the texts subsumed by each node within the tree are not shown.

CS 370 F,W 3C 0.5
Numerical Computation
Principles and practices of basic numerical computation as a key aspect of scientific computation. Visualization of results. Approximation by splines, fast Fourier transforms, solution of linear and nonlinear equations, differential equations, floating point number systems, error, stability. Presented in the context of specific applications to image processing, analysis of data, scientific modeling.
Prereq: MATH 235, 237 and one of CS 230, 246
Antireq: CS 337

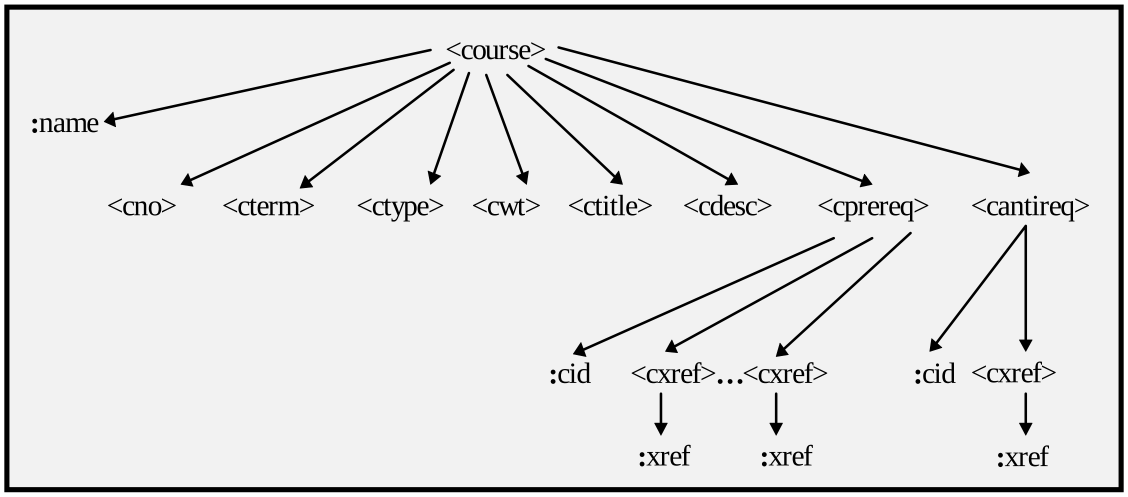
Figure 1. Part of Chapter 16 of the University of Waterloo calendar

In this sample encoding of the model, labels in the tree are "typed" as being SGML generic identifiers by the convention of using enclosing angle brackets**,** whereas attribute names are preceded by a colon. Note that in this example some SGML markup has been ignored by the modeller, as has the actual case of generic identifiers and attribute names. Other SGML types (such as entity references) can be similarly encoded, preferably preserving the convention that node types can be deduced by merely examining the first character of the associated node label. Nodes representing generic identifiers subsume the subtext within the corresponding tags, and nodes representing attribute names subsume the attribute's value. Note that not all text need be subsumed by leaf nodes: in the example, the text "and one of" is subsumed by `<course>` and `<cprereq>`, but not by any `<cxref>` (nor any other leaf).

Continuing with the above example, the four `<cxref>` course cross references that are cited as prerequisites for course CS370, may be considered to form either a list or a set. Applications that wish to treat such cross references as an unordered collection will avoid attaching unwarranted significance to the ordering of these subtexts within the above encoding. Furthermore, applications must be careful to preserve the intended semantics of the text. When considering results derived from the above encoding, for example, even though CS370 lists four prerequisites, the text states that students need not satisfy all four prerequisites prior to enrolling in CS370.

```
<COURSE NAME="CS370"><CNO>CS 370</CNO><CTERM>F,W</CTERM><CTYPE>3C</CTYPE>
<CWT>0.5</CWT><br><CTITLE>Numerical Computation</CTITLE><br><CDESC>Principles and practices of
basic numerical computation as a key aspect of scientific computation. Visualization of results. Approximation by
splines, fast Fourier transforms, solution of linear and nonlinear equations, differential equations, floating point
number systems, error, stability. Presented in the context of specific applications to image processing, analysis of
data, scientific modeling. </CDESC><br><CPREREQ CID=478><cxref xref="MATH235">MATH 235</cxref>,
<cxref xref="MATH237">237 </cxref> and one of <cxref xref= "CS230">CS 230</cxref>, <cxref
xref="CS246">246</cxref></CPREREQ><br><CANTIREQ CID=479> <cxref xref="CS337">CS
337</cxref></CANTIREQ><br></COURSE><p>
```

(a) Tagged encoding as a character string



(b) Schematic representation in the model

Figure 2. An encoding for a fragment of the calendar

The ordered hierarchical model for structured text seems an intuitive one, but may be unduly restrictive. In practice many loosely structured texts (and particularly those on the Web) violate the assumption that the markup within them is correctly nested. For example, font changes may occur at arbitrary points within a text, rather than within well-defined structural boundaries. The model allows multiple hierarchical structures to be encoded as independent substructures, but does not allow relationships between distinct structural hierarchies to be modelled directly. For example, physical page boundaries impose a secondary structure on many documents, but these physical boundaries cannot readily be related to the logical document structure within our proposed model. If the ordered hierarchical model is considered too limited, it might be possible to generalize the concept of containment and ordering used within the model, so that these concepts can be applied to overlapping regions of text (see, for example, [ISO89, Ray96b]).

3.2. Associating a schema with text

The encoding shown in Figure 2 provides a conceptual model of the structure of the text shown in Figure 1, but fails to provide key information needed by a naïve user who wishes to access this text. Such a user may have no *a priori* knowledge about the node labels present in the encoding, their interpretation, and the valid relationships between the various node labels within the text encoding. It is not possible to deduce from Figure 2 that university courses may also have corequisites associated with them, and nothing indicates if course cross references occur elsewhere within the calendar, or potentially within the course descriptions subsumed by nodes labelled `<cdesc>`. Thus, when a parser converts a string into a text tree, it associates with this text the grammar that it used when parsing it.

The schema shown in Figure 3 describes the actual information content present within Chapter 16 of the University of Waterloo calendar, and constitutes part of the schema for the calendar. Chapter 16 is partitioned into source files, each describing one or more departments. Departments have a name and associated courses. Course listings may include many details, such as descriptions, ancillary information, prerequisites, antirequisites and corequisites. Repeating elements within this schema have been marked with a '+' to improve comprehension.

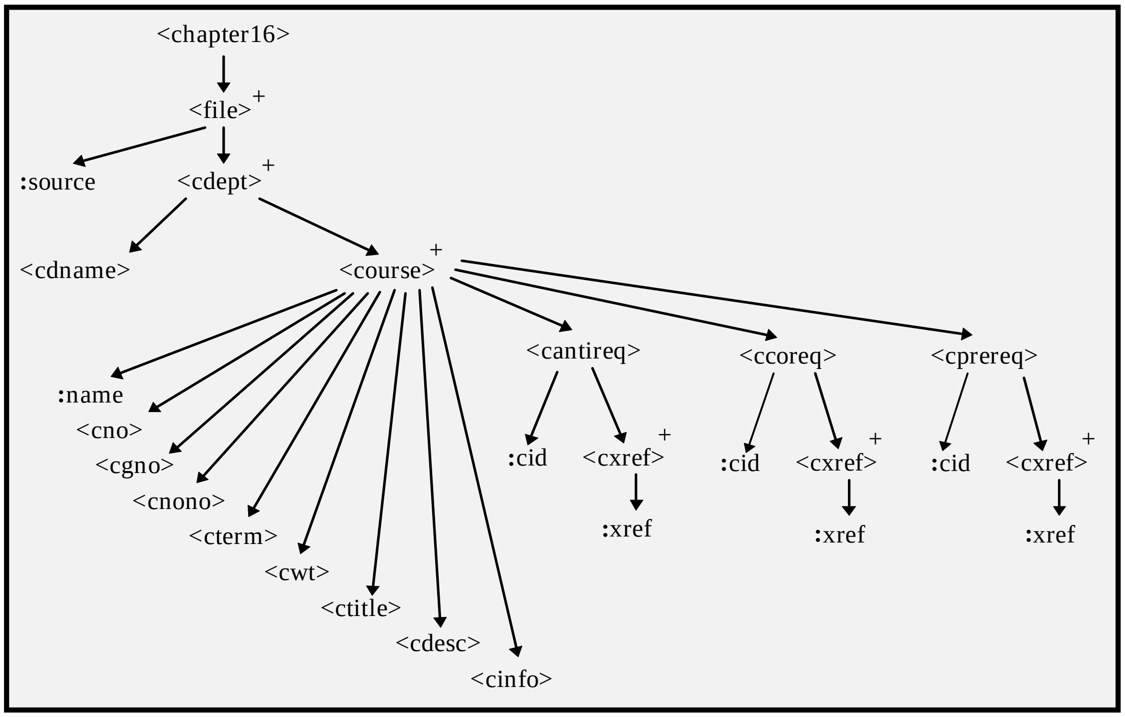


Figure 3. A schema for Chapter 16

The functions shown in Figure 4 provide rudimentary access to the schema associated with a text and enable recovery of the schematic information presented above.

Function	Returns	Description
Text_to_grammar	Grammar	The internal grammar associated with a given text
Grammar_root	Varchar	The name of the root element in the grammar
Grammar_elements	Relation	Grammar element names and informal descriptions
Grammar_hierarchy	Relation	Grammar element child/descendant relationships
Grammar_to_text	Text	Extended textual description of external grammar

Figure 4. Functions on the schema of a text

The function **text_to_grammar** returns the grammar associated with a given text. (Note that although all examples here use SGML, this is not a requirement imposed by the model.) The function **grammar_root**, when applied to such a grammar, returns the label of the root of the schema for this grammar. The function **grammar_elements**, when applied to a grammar, returns a two column relation describing each distinct node label in the schema associated with this grammar. For example, this function returns a table such as that shown in Figure 5 when applied to the calendar schema; the descriptive information forms part of the information to be provided with the parser.

Element name	Description
<chapter16>	Chapter 16
:name	Course name abbreviation
<cdept>	Department course listings
<course>	A course description
…	…

Figure 5. Part of the relation returned by grammar_elements

The function **grammar_hierarchy** when applied to a grammar returns a relation describing the transitive closure of all ancestor/descendant relationships within the grammar schema.. This function returns the table shown in Figure 6 when applied to the calendar schema.

Ancestor	Descendant	Relationship
<chapter16>	<file>	Child
<chapter16>	:name	Descendant
<chapter16>	<cdept>	Descendant
<cdept>	<cdname>	Child
…	…	…

Figure 6. Part of the relation returned by grammar_hierarchy

Additional functions can be implemented to provide further information about the schema associated by the parser to a given text. For example, none of the above functions provide information about the order of nodes within the text schema, none indicate whether ancestor/descendant relationships are optional or mandatory, and none indicate whether there is a one-to-one or one-to-many relationship.

The **grammar_to_text** function is not the inverse of the **text_to_grammar** function, but instead produces a structured text (i.e., a value of type text that can itself be operated on as a text tree). This text further describes the internal grammar associated with the text. If no such text exists the function returns null. For example, the text of the grammar associated with an SGML document might include a model of the actual document type definition (DTD) used when parsing and validating this SGML document. Although the method used to encode a DTD might be very different from the method used to encode the original SGML text, providing a textual representation of the grammar allows the full power of the proposed text extensions to be employed not only against an arbitrary structured text, but against an arbitrary textual description of the grammar associated with a text.

It should be stressed that the above approach associates a grammatical schema with every instance of text rather than merely with a collection of texts residing in a single relational column or belonging to a particular set of textual objects. Based on industrial practice, it is unrealistic to demand that only texts having exactly the same grammatical schema can be grouped into collections. However, because it is difficult to perform set-at-a-time operations against collections of text that share little in common, applications may choose to impose constraints on text collections to ensure that the grammars of all contained texts share certain features.

3.3. Marking structured text

Previous proposals [ATA91] have recognized the importance of allowing fragments of subtext to be marked so that, for instance, these fragments may be highlighted when viewed. In environments that support update and storage of marked subtexts, such marks may also be used to store the state necessary to support interactive hierarchical text navigation and browsing, through a stateless SQL interface.

To allow fragments of structured text to be marked, the structured text model described above is extended so that any node within this model may be either marked or unmarked. Thus, in the model an instance of structured text

1. spans a given region of text,
2. has an identifiable grammar associated with it, and
3. includes a set of zero or more marks that identify specific structured subtexts within this text.

Two texts that span exactly the same region of text and share the same grammar, but may have different marks, are said to share the same *provenance*.

All of the functions shown in Figure 7 take one or more input texts that share the same provenance and return a new text that also shares this same provenance. These functions allow marks within a structured text to be manipulated.

Function	Returns	Description
Mark_subtexts	Text	Mark subtexts in texts using a hierarchical pattern
Union_marks	Text	Union marks in two instances of the same text
Intersect_marks	Text	Intersect marks in two instances of same text
Except_marks	Text	Compute the set difference between two sets of marks
Keep_marks	Text	Keeps pre-ordered marks in a given range within a text
Aggregate_marks	Text	Union marks in grouping of the same instance of text

Figure 7. Functions that manipulate marks in a text

The function **mark_subtexts** takes as input an instance of text and a string containing instructions about how the resulting text is to be marked (*cf*. [Kil93]). The structured text pattern matching language used to encode these instructions within the string is presented using the BNF for <*pattern*> in Figure 8. Figure 9 illustrates how text nodes are matched against a structured text pattern, assuming the schema from Figure 3. (Note that '%' matches zero or more consecutive characters within a text label.) An alternative syntax using more descriptive function names rather than the compact notation presented here has also been defined.

```
<pattern>          ::= <node_rule> [ <descendants> ] | <node_rule>
<descendants>      ::= <set> | <list>
<set>              ::= <pattern> <ampersand> <set> | <pattern>
<list>             ::= <pattern> <comma> <list> | <pattern>

<node_rule>        ::= <rooted_rule>
<rooted_rule>      ::= <rooted> <marked_rule> | <marked_rule>
<marked_rule>      ::= <marked> <marking_rule> | <marking_rule>
<marking_rule>     ::= <node_pattern> <flagged> | <node_pattern>
<node_pattern>     ::= <node_label> { <text_expression> }    | <node_label>

<node_label>       ::= <characters>
<text_expression>  ::= <characters>
<characters>       ::= <characters> <character> | <character>
<character>        ::= !! Any appropriately escaped character !!

<ampersand>        ::= &
<comma>            ::= ,
<rooted>           ::= ^
<marked>           ::= @
<flagged>          ::= #
```

Figure 8. The structured text pattern matching language

Pattern	Marks
`%#`	Every node in the text
`^%#`	The root of the text
`^%[^%#]`	Every child under the root of the text
`@<course>#[<cprereq>]`	Every marked course having prerequisites
`<course>#[<cprereq>[:xref{CS370}]]`	Courses that list CS370 as a prerequisite
`<chapter%>#[<file>&<file>]`	Chapters containing more than one file
`<course>#[<cwt>,<cterm>]`	Courses whose weight appears before term
`<course>[:name{CS370}&<cwt>#]`	The course weight of CS370
`@%[^<cxref>#]`	Every cxref that is a child of a marked node
`%[:source#,<ctitle>#,<cdesc>#]`	Every in order occurrence of source, ctitle and cdesc

Figure 9. Examples of how structured text patterns match a text

The <*pattern*> and <*descendants*> productions in Figure 8, allow a simple one- dimensional representation of a partially ordered pattern tree to be expressed. Within this expression, each <*pattern*> within a <*list*> (e.g., *B*, *C*, and *D* in the pattern '*A*[*B*,*C*,*D*]') constitutes an ordered descendant of the <*node rule*> immediately preceding this <*list*> within the pattern, and each <*pattern*> within a <*set*> constitutes an unordered descendant of the immediately preceding <*node rule*>.

The structured text pattern matches a subset of the nodes in an instance of structured text when
(a) every <*node rule*> is associated with exactly one distinct node in the structured text,

(b) every ancestor/descendant relationship between <*node rule*>s in the structured text pattern holds between the corresponding matched nodes within the text,
(c) ordered lists of nodes within the pattern appear in the same order as the nodes that they match within the text,
(d) any <*node rule*> containing the <*rooted*> symbol matches a node within the text whose parent node (if any) is also simultaneously matched by a corresponding <*node rule*>,
(e) each <*node rule*> containing the <*marked*> symbol matches a marked node within the text,
(f) the <*node label*> agrees with the corresponding node label within the text, and
(g) the text subsumed by a matched node satisfies the <*text_expression*> (if present).

The rules governing how node labels and subsumed text are matched against strings within the pattern tree depends on the environment within which the structured text abstract type is supported. Within SQL it is proposed that a <*node label*> use the symbols '%' and '_' as wildcards, that this <*node label*> be compared with structured text labels using the SQL 'like' predicate [ISO92], and that this comparison be case insensitive. It is proposed that the <*text_expression*> be a valid SQL/MM 'contains' clause [ISO96s]; when applied against the subsumed text, it identifies structured text nodes matching this expression. As a possible extension, it might be better to allow <*text expression*> to be an arbitrary SQL predicate (potentially containing more than just a Full Text search specification); this would increase the power of the pattern matching language considerably, and it might simplify detecting cases where certain complex text operations could be optimized.

Because chain patterns are commonly used in text searching, the pattern matching language is extended with two syntactic shorthands: `A..B` represents an ancestor-descendant relationship (equivalent to `A[B]`), and `A.B` represents a parent-child relationship (equivalent to `A[^B]`).

The function **mark_subtexts** identifies each possible matching (if any) between nodes in the input text and the structured text pattern, and marks any node within the matched text that corresponds to a <*node rule*> containing the <*flagged*> symbol.

The functions **union_marks**, **intersect_marks** and **except_marks** take as input two instances of text with the same provenance and return a new text of that same provenance having marks that are respectively the union/intersection/set difference of the marks in the input texts. For example:

```
intersect_marks(
  mark_subtexts(calendar, '<course>#[<cprereq>[:xref{CS370}]]'),
  mark_subtexts(calendar, '<course>#[<cwt>,<cterm>]')
)
```

marks courses in the calendar that have CS370 as a prerequisite and list the course weight before the term in which the course is offered.

The function **keep_marks** takes as input a text and an integer range (expressed as a start position and a length). Marks in the input text are assigned ordinals (the first such being one) consistent with the order that they would be visited in by a pre-order traversal of the text tree, and those marks within the input text (if any) having ordinals lying in the specified range are the only marks preserved in the resulting text. The function **aggregate_marks** takes as input a collection of texts having the same provenance, and returns a new instance of text having this provenance and containing the union of all marks in the collection of input texts. The functions **union_marks**,

intersect_marks, **except_marks**, and **aggregate_marks** raise an appropriate exception when their input texts do not share the same provenance.

Marking optional subtexts proved challenging and inefficient. Optional subtexts cannot be marked concurrently with mandatory subtexts, since the tree pattern matching language is based on performing an exact match against all described subtexts. Optional subtexts are therefore marked (and extracted, as described below) in a second phase after mandatory subtexts had been identified. This is elaborated at the end of the next section.

In the current SQL/MM standard [ISO96s], there is no ability to use structured text concepts or marking within a Full Text search specification. At present the Full Text specification uses the concepts of character, word, sentence and paragraph within its own search language, without defining or explaining how such concepts relate to the actual material contained within an arbitrary instance of Full Text. This problem could be resolved by viewing these concepts as specific instances of well-defined structure associated with the text being searched and augmenting the SQL/MM Full Text specification so that it allows marking of identified substrings matching Full Text patterns and searching that incorporates structured text concepts. Ideally, full interplay should be allowed between “horizontal” and “hierarchical” text searching and marking, thus making the resulting language much more expressive. This would also allow the concept of proximity, which is well defined within the Full Text proposal, to be equally effective within our structured text proposals.

3.4. Extracting structured subtext

Each of the functions shown in Figure 10 extracts from an input text a collection of subtexts, returning a relation that contains the extracted subtexts.

Function	Returns	Description
Isolate_subtexts	Relation	Extracts all marked subtexts within a text
Extract_subtexts	Relation	Extracts the subtexts that match the specified pattern

Figure 10. Functions that extract subtexts from a text

The function **isolate_subtexts** takes an instance of text as input and, for each mark within this text, produces an output row within the resulting relation. The first attribute within this output row contains an instance of text having the same provenance as the input text, but having only the solitary mark within this text that caused the row to be generated. The second attribute in the row contains, as a new instance of text, the subtext rooted at this mark. The mark on the root is removed from the resulting subtext, but all other marks within the resulting subtext are preserved.

The function **extract_subtexts** takes as input an instance of text and a structured text pattern as described for **mark_subtexts** above, and it produces a relation with one row for every possible complete match, as described below. The number of columns in the resulting relation depends on the text pattern. In environments where this value must be known at compile time, a third parameter indicating the expected number of columns in the resulting relation must be included. In environments such as SQL2, for example, this third argument must be an integer constant, and the function **extract_subtexts** will raise an appropriate exception if the resulting relation does not contain exactly the number of columns indicated.

Let the number of flagged <*node rule*>s in the pattern be *n*. For every distinct method of matching the *n* flagged <*node rule*>s within the structured text pattern against nodes in the structured text, while concurrently matching in at least one way the entire structured text pattern against the text, an output row is produced with *n+1* columns. The first column contains a new text with the same provenance as the input text, while the remaining *n* columns contain the subtexts that matched the flagged <*node rule*>s, in the left to right order (pre-order) that they occurred within the structured text pattern. Each subtext remains marked in the innermost extracted ancestor within this tuple. No other marks are present in the texts contained with the output tuple.

For example, if the operation:

```
    extract_subtexts(calendar, 3, '<course>[:name#, <cxref>..:xref#]')
```

is applied to the subtext shown in Figure 2 the relational rows shown in Figure 11 are returned in no specific order. Within this figure marked subtexts within a text are shown in bold.

<course **name="CS370"**... ... <cxref **xref="MATH235"**>MAT ... <p>	name="CS370"	xref="MATH235"
<course **name="CS370"**... ... <cxref **xref="MATH237"**>237< ... <p>	name="CS370"	xref="MATH237"
<course **name="CS370"**... ... <cxref **xref="CS230"**>CS 230< ... <p>	name="CS370"	xref="CS230"
<course **name="CS370"**... ... <cxref **xref="CS246"**>246< ... <p>	name="CS370"	xref="CS246"

Figure 11. Result returned by extract_subtexts

This is a very different result from that shown in Figure 12 returned by:

```
   isolate_subtexts(
        mark_subtexts(calendar,'<COURSE>[:NAME#, <CXREF>..:XREF#]')
   )
```

<course **name="CS370"** ... xref="MATH235">MAT ... <p>	name="CS370"
<course name="CS370" ... **xref="MATH235"**>MAT ... <p>	xref="MATH235"
<course name="CS370" ... **xref="MATH237"**>237< ... <p>	xref="MATH237"
<course name="CS370" ... **xref="CS230"**>CS 230< ... <p>	xref="CS230"
<course name="CS370" ... **xref="CS246"**>246< ... <p>	xref="CS246"

Figure 12. Result returned by isolate_subtexts

The ability to extract a subtext while preserving the context within which it was extracted is significant. This avoids information loss, and allows aggregation of subtexts back into the text from which they were earlier extracted. Unfortunately, since the context is preserved in the *containing* text (by marking those subtexts extracted from this text), it becomes difficult to preserve context when multiple concurrent extractions are performed against a single instance of text. This is because it is difficult to determine which mark within the containing text corresponds to which instance of extracted subtext. For example, in the first column of the first tuple in Figure 11, two subtexts are marked; which mark belongs to the text in the second column and which to the text in the third? In this case, the correspondence is easy to determine, but if the pattern used '&' in place of ',' the matches could occur in either order in the text instance and the extracted texts may not be so simple to distinguish from each other. To address this problem it is proposed that the

<*flagged*> production shown in Figure 8 be augmented so that a second '#' be allowed to immediately follow the first. Subtexts extracted as a result of a '##' operator would be immediately preceded (within the output relation) by a column containing the original text in which only this subtext was marked.

As mentioned at the end of the previous section, extracting optional subtexts proved challenging and inefficient. Optional subtexts must be extracted in a second phase after mandatory subtexts had been identified. The mandatory and optional subtexts are subsequently related through the use of an appropriately constructed outer join, and absent subtexts are represented within such an extraction process by null. This second extraction phase is very inefficient since it is applied separately to each grouping of mandatory subtexts within a single tuple, rather than being applied during the construction of these distinct tuples. In addition, the division of subtext extraction into multiple independent phases makes it difficult to enforce contextual relationships between mandatory and optional subtexts that otherwise would have been readily expressible within the structured text pattern matching language. Extensions to the proposed pattern matching language that would provide support for optional matching of text are being considered (*cf.* optional matching in the context of specific semi-structured data in OEM-QL [Pap95]).

More generally, one often wants to recover structured text that approximates, but does not exactly match, the search specification provided. There is a need to be able to compute how well instances of subtext match a given search specification as a ranking, and to recover from such matchings (in a suitable order) those subtexts that exceed some specified ranking threshold. Such a facility would readily allow support for optional subtext matching, since such optional matchings could be assigned a small (possibly zero) weight within the overall ranking scheme.

3.5. Other text operations

The functions shown in Figure 13 perform a variety of operations that complete the description of the structured text abstract data type.

Function	Returns	Description
String_to_text	Text	Parses the input string using a specified method
Text_to_string	String	Converts text to a string using a specified method
Text_match	Boolean	Matches text against a hierarchical tree pattern
Count_marks	Integer	Counts the number of marks in a text
Cast	?	Directly casts a text to an integer/double/date etc.

Figure 13. Other functions associated with structured text

The function **string_to_text** takes as input two strings. The first string contains the text to be parsed and the second contains a keyword identifying how this text is to be parsed (i.e., which parser and which grammar to apply). If the input string is successfully parsed, the function returns the corresponding instance of structured text, conforming to the model used by the parser. Complementing this, the function **text_to_string** produces a string from a text. A choice of conversion methods is provided, since text can be linearized and presented in many ways. For example, one converter may produce a tagged string, a second might omit all tags, and a third might suppress particular subtexts.

The function **text_match** accepts the same inputs as **mark_subtexts**, but rather than marking texts, it merely returns true if the pattern matches the text in at least one way. The function **count_marks** takes as its input an instance of text and returns the number of marked nodes within this text.

Within our prototype suitably encoded texts (*cf*. [Gon87]) can be directly **cast** into numeric integers, double precision values and dates, and they can be efficiently recovered using their “external” representation. This allows large relations to be directly encoded within a text while continuing to be rapidly accessible.

4. A sample application

The University of Waterloo undergraduate calendar provides a considerable amount of textual information about events, courses, awards, faculty members, departments and university regulations. Each year this document is marked up using HTML and made available on the World Wide Web [UW96].

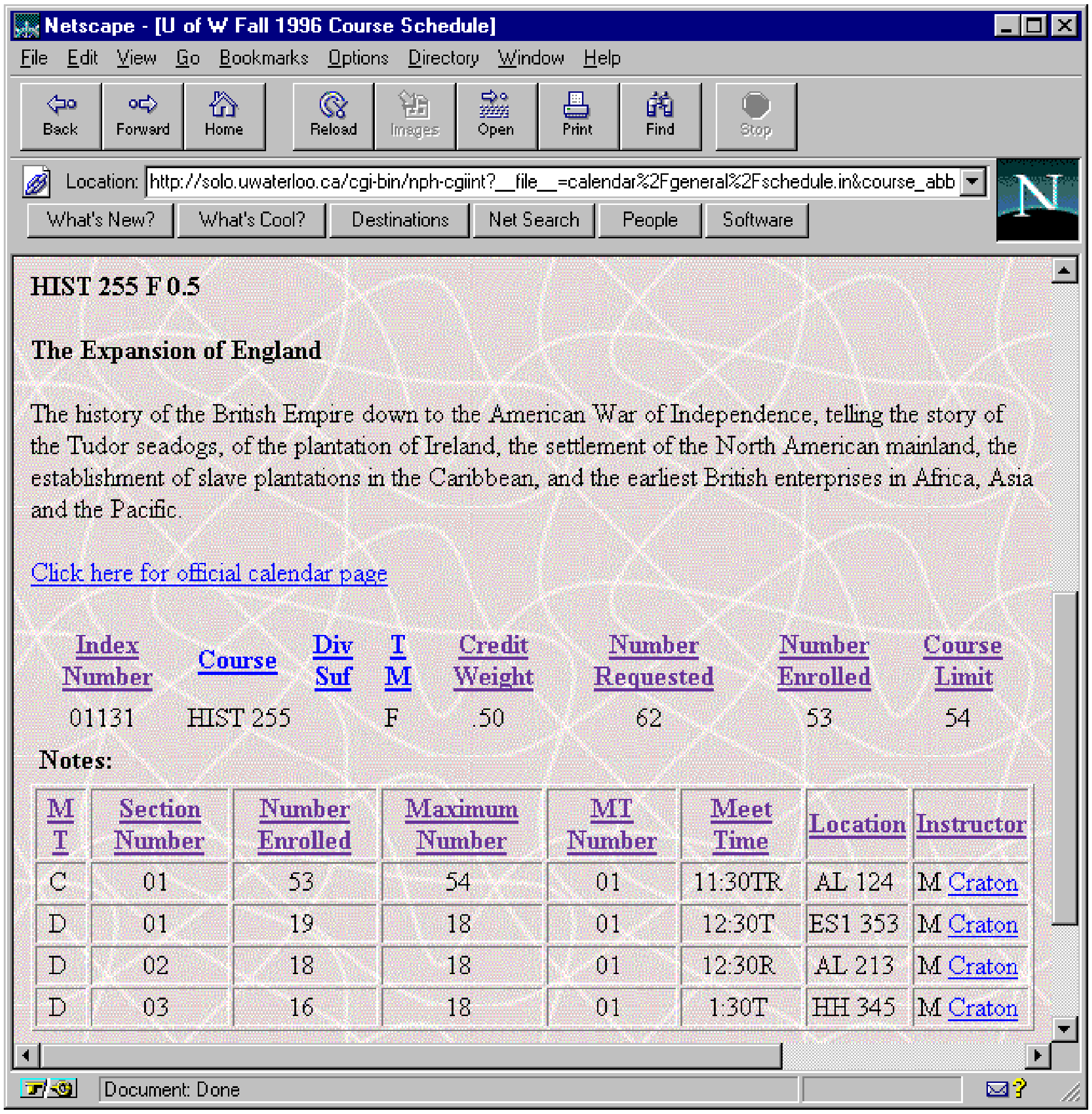
While some benefits result from making the raw material contained within the calendar available on the Web, locating desired information within the calendar is often difficult, since large volumes of text must be visually scanned, and few facilities exist to relate complementary information within the calendar. Summary information can only be derived by examining all relevant sections of the calendar mechanically, and relationships between the calendar and alternative sources of information cannot be exploited.

We addressed the above limitations by developing a prototype web application that provides alternative methods of accessing the calendar [UW97]. After adding appropriate descriptive SGML markup to the calendar (as shown in Figure 2a), the resulting document was indexed so that it could be rapidly searched by Open Text’s text search engine. Front end Web applications were built to demonstrate how context specific information can be retrieved by our hybrid query processor, which also provided simultaneous access to additional resources (including course schedule and personnel tables) stored in an Oracle database.

Those responsible for maintaining the calendar derived immediate benefit from having the data loaded into a database. Since we required that our input source texts conform to HTML, we encouraged corrections in HTML pages that might otherwise have caused client browsers to fail. In developing an extended DTD describing the descriptive structure associated with the various sections within the calendar, we formalized the previously implicit rules governing how various departments prepare material for inclusion within the calendar, and as a result moved closer to standardizing and automating the data entry process associated with construction of a yearly calendar.

Having added descriptive markup to the text, it became possible to validate textual information contained within the calendar more easily. It is, for example, easily possible to extract from the calendar the names, office locations and phone numbers of all members of faculty listed as the contact people for information relating to courses. This information can be validated against corresponding information in a current telephone directory stored within an Oracle database. If desired, relational information derived from the calendar can even be imported directly into conventional relational database systems for use in alternative applications.

Students derived immediate benefits from being provided with improved access to the University of Waterloo calendar. One student who was particularly interested in courses relating to Ireland was able to discover immediately that History 255 “The Expansion of England” was the only course within the calendar to include the word Ireland within its course description, and he was then able to recover the course schedule associated with this course. Figure 14 shows the screen output, with the course description for History 255 at the top of the screen matched with the corresponding course schedule information selected from relational tables shown below.


Netscape - [U of W Fall 1996 Course Schedule]
File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Location: http://solo.uwaterloo.ca/cgi-bin/nph-cgiint?__file__=calendar%2Fgeneral%2Fschedule.in&course_abb
What's New? What's Cool? Destinations Net Search People Software
Document: Done

HIST 255 F 0.5

The Expansion of England

The history of the British Empire down to the American War of Independence, telling the story of the Tudor seadogs, of the plantation of Ireland, the settlement of the North American mainland, the establishment of slave plantations in the Caribbean, and the earliest British enterprises in Africa, Asia and the Pacific.

Click here for official calendar page

Index Number	Course	Div Suf	T M	Credit Weight	Number Requested	Number Enrolled	Course Limit
01131	HIST 255		F	.50	62	53	54

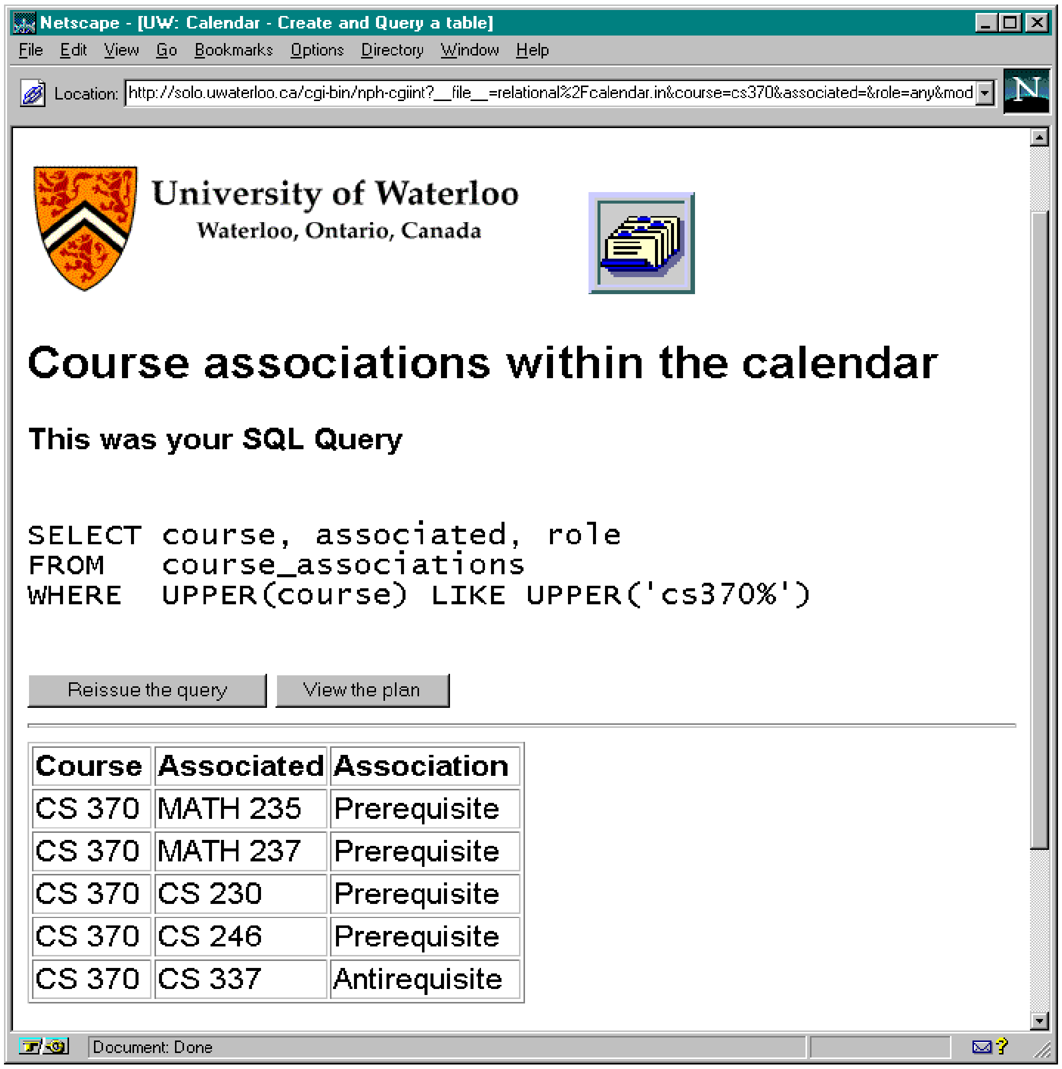
Notes:

M T	Section Number	Number Enrolled	Maximum Number	MT Number	Meet Time	Location	Instructor
C	01	53	54	01	11:30TR	AL 124	M Craton
D	01	19	18	01	12:30T	ES1 353	M Craton
D	02	18	18	01	12:30R	AL 213	M Craton
D	03	16	18	01	1:30T	HH 345	M Craton

Figure 14. Output that relates structured text with relational data

Members of faculty and administrators also found uses for the resulting system. It is, for example, possible to identify all members of faculty within the university who hold one or more degrees from specific universities, have specific positions, belong to specific departments, and/or perform given administrative roles. Such queries can also be supported by structured text engines, such as PAT, directly. However, it is possible to perform very much more complex queries using the inherent expressive power of SQL2, if as an end user or application designer one is capable of formulating the necessary SQL queries.

The Registrar's office had long wanted to validate the relationships that exist between course descriptions, but it had been previously unable to derive tables that summarize the relationships between a course description and its internally documented prerequisites, corequisites, and antirequisites. Upon learning of this, a relational view `course_associations`, capturing all described course pairings, was easily defined using the extended text operators, and it was quickly added to our demonstration and made available for use by members of the Registrar's office and others (Figure 15).


Netscape - [UW: Calendar - Create and Query a table]
File Edit View Go Bookmarks Options Directory Window Help
Location: http://solo.uwaterloo.ca/cgi-bin/nph-cgiint?__file__=relational%2Fcalendar.in&course=cs370&associated=&role=any&mod
University of Waterloo
Waterloo, Ontario, Canada
Reissue the query
View the plan
Document: Done

Course associations within the calendar

This was your SQL Query

```
SELECT  course, associated, role
FROM    course_associations
WHERE   UPPER(course) LIKE UPPER('cs370%')
```

Course	Associated	Association
CS 370	MATH 235	Prerequisite
CS 370	MATH 237	Prerequisite
CS 370	CS 230	Prerequisite
CS 370	CS 246	Prerequisite
CS 370	CS 337	Antirequisite

Figure 15. Presenting course associations as a relation

The Student Awards office asked us to provide access to financial award information contained within the calendar and were pleasantly surprised to discover that the necessary work of marking them had been completed prior to their request. End users were therefore already able to search for awards, grants and scholarships, using various criteria, including numeric considerations associated with an award. It is possible, for example, to select awards that cite some maximum, minimum, average or total set of award amounts within them, or that include award amounts in (or not in) a given numeric range. The importance of accessing text through SQL is evident here, since complex numeric processing and aggregation is typically not supported by existing text search engines.

The Faculty of Mathematics was asked to provide information about the number of members of faculty at different ranks by department, and to correlate this information against the number of courses, and if possible, students taught. It was easy using SQL to derive a table from the calendar that documented the number of members of faculty at various ranks by department. The courses taught by a department in a particular term, and the enrollment in these courses could be as readily obtained from the course schedule information stored in the Oracle database, and this collective information could be immediately integrated into the desired relational tables, by using the ability of the hybrid query processor to join relations from distributed data sources.

5. Conclusions

This paper has described an abstract data type for structured text that can readily be incorporated into existing text searching technology, object database technology or forthcoming SQL3 technology. This abstract data type can be used to perform complex text- and relational-intensive queries in widely distributed heterogeneous environments, such as those rapidly appearing on the World Wide Web.

Our text extensions have proven highly effective in allowing structured text to be queried, retrieved, and integrated with relational information. The concept of allowing selected subtexts within a text to be marked is a natural one, and it is powerful when coupled with set-at-a-time processing, facilities to extract subtexts, and further pattern matching operations.

The proposed text extensions allow easy definition and dynamic construction of relational views of structured text derived from hierarchically structured text, marked subtexts, and/or extracted subtexts. This allows naturally occurring relations within text to be easily retrieved, without requiring that the text itself be stored within a relational system. Thus diverse relational views can be superimposed on portions of the text without imposing a single “master” relational view on the whole text. The use of a high-level, non-procedural text pattern matching language simplifies the definition and construction of such relations, while facilitating encapsulation and optimization of the software responsible for integrating text and relational data. As a result, text can be retained in its original form and still be subjected to expressive database operators.

The software we have implemented to support the structured text model performs well when accessing both text and relational data. It has been used to construct a moderately sophisticated suite of Web-based applications that allows integration of information contained within the text of various chapters of the University of Waterloo Undergraduate calendar with course schedules, phone lists, and other tabular data stored in relational databases. The same system also provides relational access to other structured texts, including *The Oxford English Dictionary*, *The Collected Works of Shakespeare*, *The Devil’s Dictionary* and *The Bible* [UW97].

The described SQL structured text extensions have been accepted for inclusion within the evolving SQL/MM standard [Dav96]. They are of immediate benefit to any user who wishes to integrate textual information into their existing relational database systems, and to any user currently involved in text intensive searching or querying who wished to capitalize on the expressive power of SQL. The text abstract data type is also suitable for inclusion in object-oriented database systems. These structured text extensions are simple ones that can be easily understood, and yet are surprisingly effective in selectively recovering and consolidating relevant information from within the very complex structures that occur naturally within many types of text. Thus, our experiences in designing and implementing these text extensions should prove valuable to those who wish to extend relational and object-oriented systems so that they accommodate structured text.

Our research is also of immediate benefit to text engine vendors, since it provides a very easy method of integrating text engine technology with both SQL2 and SQL3. We have shown that it is feasible to implement relational wrappers for several text search engines to extend relational database systems so that they provide support for complex text extensions. We have also shown that it is possible to integrate such extensions efficiently into SQL, so that vendor-specific objects may be rapidly retrieved and manipulated using standard SQL constructs. Furthermore, we demonstrated how the integrated text-relational technology can be further integrated with Web technology.

Acknowledgments

This work has been carried out as part of the University's participation in the Canadian Strategic Software Consortium (CSSC), which also includes Fulcrum Technologies Inc., Grafnetix Systems Inc., InContext Corporation, Megalith Technologies Inc., Open Text Corporation, Public Sector Systems, and SoftQuad Inc. CSSC was formed in 1993 to perform pre-competitive research on the integration of relational and text databases and was partially supported by Industry Canada's Strategic Technologies Program (STP).

Ideas expressed in this paper have been developed and refined in part through discussions with members of the CSSC's Hybrid Query Processor (HQP) working group. Special acknowledgment is due to past contributors to the Text/Relational Database Management Systems project, including Betty Blake, Gaston Gonnet, Pekka Kilpeläinen, Eila Kuikka, Paul Larson, and Tim Snider. Financial assistance was provided by the University of Waterloo and through grants from the Natural Sciences and Engineering Research Council of Canada, Industry Canada, and Open Text.

References

[ATA91] Air Transportation Association, *Advanced Retrieval Standard – SFQL: Structured Fulltext Query Language*. ATA-89-9C SFQL Committee, ATA specification 100, Rev 30, Version 2.2, Prerelease C, October 1991, 84 pp.

[Bla94] G.E. Blake, M.P. Consens, P. Kilpelainen, P-Å. Larson, T. Snider, and F.W. Tompa, "Text/Relational Database Management Systems: Harmonizing SQL and SGML," *Proc. Application of Databases (ADB 94).*, Vadstena, Sweden (June 1994), *Lecture Notes in Computer Science 819*, Springer-Verlag, pp. 267-280.

[Bla95] G.E. Blake, M.P. Consens, I.J. Davis, P. Kilpelainen, E. Kuikka, P-Å. Larson, T. Snider, and F.W. Tompa, *Text/Relational Database Management Systems: Overview and Proposed SQL*

Extension. University of Waterloo Department of Computer Science Technical Report CS-95-25 (June 1995).

[Bri97] M. Brisebois and I.J. Davis, “HQP: la gestion et l’intégration des données relationnelles et textuelles,” *L’expertise informatique 3*, 1 (été 1997) pp. 8-13.

[Cla94] C.L.A. Clarke, G.V. Cormack, and F.J. Burkowski, *Fast Inverted Indexes with Online Update*. University of Waterloo Department of Computer Science Technical Report CS-94-40 (November 1994) 11 pp. See also *http://multitext.uwaterloo.ca.*

[Cob92] N. Coburn and P-Å. Larson, “Multidatabase Services: Issues and Architectural Design,” *Proc. 1992 CAS Conf. (CASCON),* IBM, pp. 57-66.

[Coo87] J.H. Coombs, A.H. Renear, and S.J. de Rose, “Markup Systems and the Future of Scholarly Text Processing,” *Comm. ACM 30*, 11 (November 1987) pp. 933-947.

[CSSC94] *CSSC News Letter*. Issue 1, December 19, 1994.

[Dav96] I.J. Davis, *Adding structured text to SQL/MM Part 2: Full Text*, A change proposal. ISO/IEC JTC1/SC21/WG3 CAC N334R3, April 26, 1996.

[Fla96] D. Flanagan, *Java in a Nutshell,* O’Reilly and Associates, 1996.

[Ful94] Fulcrum Technologies Inc., *Fulcrum SearchServer Version 2.0: Introduction to SearchServer*, 1994.

[Gon87] G. H. Gonnet, “Extracting information from a Text Database. An example with dates and numeric data,” *Proc. Third Conf. UW Centre for the New Oxford English Dictionary*, Waterloo, Canada (November 9-10, 1987) pp. 89-96.

[ISO86] International Organization for Standardization, *Information processing - text and office systems - Standard Generalized Markup Language (SGML)*. ISO 8879: 1986.

[ISO89] International Organization for Standardization, *Information processing - text and office systems - Office Document Architecture (ODA)*. ISO 8613-2: 1989.

[ISO92] International Organization for Standardization, *Information technology - Database languages - SQL*. ISO/IEC 9075: 1992.

[ISO96d] International Organization for Standardization, *Document Style Semantics and Specification Language*. ISO/IEC 10179:1996, http://www.jclark.com/dsssl.

[ISO96s] International Organization for Standardization, *SQL Multimedia and Application Packages. Part 2: Full Text*. ISO/IEC Working Draft, June 1996.

[Kil93] P. Kilpeläinen and H. Mannila, “Retrieval from hierarchical texts by partial patterns,” *Sixteenth Int. ACM SIGIR Conf. on Research and Development in Information Retrieval* (1993) pp. 214-222.

[Mac92] I.A. Macleod, “Data Modelling Requirements for Document Management,” *Proc. IFIP TC8/WG8.1 Working Conference on Information System Concepts: Improving the Understanding*, Alexandria, April 1992, Elsevier (North-Holland) pp. 259-271.

[Mic92] *Microsoft ODBC 2.0 Programmer’s Reference and SDK Guide*. Microsoft Press. 1992

[OTC95] Open Text Corporation, *Open Text 5 System Integration Guide and Database Administration Guide*, 1995.

[Pap95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom, “Object Exchange Across Heterogeneous Information Sources,” *Proc. Eleventh Int. Conf. on Data Engineering*, Taipei, Taiwan (March 1995) pp. 251-260.

[Rag97] D. Raggett, *HTML 3.2 Reference Specification*, The World Wide Web Consortium, REC-html32, January 14, 1997 (*http://www.w3.org/TR/REC-html32.html*).

[Ray96a] D.R. Raymond, F.W. Tompa, and D. Wood, “From Data Representation to Data Model: Meta-Semantic Issues in the Evolution of SGML,” *Computer Standards and Interfaces 18* (1996) pp. 25-36.

[Ray96b] D.R. Raymond. *Partial Order Databases*. University of Waterloo Department of Computer Science Technical Report CS-96-01 (March 1996).

[Sal96] A. Salminen and F. W. Tompa. *Grammars++ for Modelling Information in Text*. University of Waterloo Department of Computer Science Technical Report CS-96-40 (November 1996), 46 pp.

[Spe94] C.M. Sperberg-McQueen and L. Burnard (eds.), *Guidelines for the Encoding and Interchange of Machine-Readable Texts (TEI P3).* Assoc. for Computing in the Humanities, Assoc. for Computational Linguistics, and Assoc. for Linguistic and Literary Computing, April 1994 (*http://www.uic.edu/orgs/tei/p3/*).

[Tom89] F.W. Tompa, “What is (tagged) text?” *Dictionaries in the Electronic Age: Proc. 5th Conf. of University of Waterloo Centre for the New OED*, Oxford, UK (September 1989) pp. 81-93.

[UW96] University of Waterloo, *1996-97 Undergraduate Calendar,* Office of the Registrar, (*http://www.adm.uwaterloo.ca/infoucal*).

[UW97] University of Waterloo, *The TRDBMS project: Integrating structured text and SQL*, *http://solo.uwaterloo.ca/trdbms/index.html*, Department of Computer Science, 1997.

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

[Zhu92] Q. Zhu, “Query Optimisation in Multidatabase Systems, *Proc.1992 CAS Conference (CASCON)*, IBM, pp. 111-127.

Appendix A
Behind the scenes

Introduction

This appendix contains two complete queries illustrating the use of the structured text abstract data type within the context of SQL. These queries operate against the University of Waterloo calendar [UW96]. The calendar text is stored within a one row table named *uwcalendar* containing a single column named *calendar*. This table is accessed through PAT, with the aid of a relational wrapper.

Query 1

List professors and their departments for professors who have some degree from Toronto and an MBA from any institution.

Within the calendar text, the faculty is listed by department, as in the following snapshot:

Accounting

Professor, Director, School of Accountancy
J.H. Waterhouse, *BSc, MBA (Alberta), PhD (Washington, Seattle)*

Associate Professor, Acting Director, Director Professional Programs, Gordon H. Cowperthwaite Professor of Accounting
H.M. Armitage, *BSc (McGill), MBA (Alberta), PhD (Michigan State), CMA, FCMA*

Professor, Graduate Officer, The Ontario Chartered Accountant's Chair in Accounting
G. Richardson, *BA (Toronto), MBA (York), PhD (Cornell), CA, FCA*

Associate Professor, Undergraduate Officer
D.T. Carter, *BComm, MBA (Windsor), CA, FCA*

…

In the model for the calendar text, the department name is subsumed by by a node labelled `<FDNAME>`, the department members are subsumed by a node labelled `<FGRP>`, the information for each professor is under a node labelled `<FP>`, and his/her degrees are under a single node labelled `<FQUAL>`.

```
SELECT  TEXT_TO_STRING (prof_info,'clear'),  TEXT_TO_STRING (dept_name, 'clear')
FROM    (SELECT  UNNEST
                EXTRACT_SUBTEXTS(
                    calendar,
                    3,
                   '<file>[<FDNAME>#&<FGRP>[<FP>#[<FQUAL>["Toronto"&"MBA"]]]]'
                )
         FROM   uwcalendar
         ) T1(marked_calendar, dept_name, prof_info)
WHERE   prof_info IS NOT NULL
```

The keyword *unnest* (in the nested *select*) represents a proprietary extension to SQL, which allows projected functions that return relational tables to be unnested [Bla95]. Within SQL3 it has been proposed that such an operation would be replaced by one performing a left join on a table containing the inputs to the projected function, with the specific function. For this to be a viable method of performing the desired operation, the scope in which variables are known has to be extended so that inputs on the left of a join remain visible to functions used in producing the right component of the join. It is also necessary that such a correlated join implicitly join each row produced by the left input with all rows derived from this left row's inputs.

Using this alternative construction, the *unnest* would be written as:

```
SELECT      marked_calendar, dept_name, prof_info
FROM     (
      (SELECT calendar FROM uwcalendar)
            LEFT JOIN
            EXTRACT_SUBTEXTS(calendar, 3, '<file>[<FDNAME>… ]]]]' )
      ) T(calendar, marked_calendar, dept_name, prof_info)
```

Result 1

'G. Richardson BA (Toronto), MBA (York), PhD (Cornell), CA, FCA'	'Accounting'
'W.M. Lemon BA (Western Ontario), MBA (Toronto), PhD (Texas at Austin), CA, FCA, CPA'	'Accounting'
'W.D. Poole BA (Toronto), MBA (York), MSc (London)'	'Drama and Speech Communication'
'J.H. Bookbinder MBA (Toronto), MS, PhD (California, San Diego)'	'Management Sciences'

Query 2

The second example presents the SQL query used to produce the Web page shown in Figure 14. In this query, course schedules (located in an Oracle database as *schedule_courses*) are joined with the course sections for that course (also located in an Oracle database as *schedule_sections*). Then the appropriate course descriptions extracted from the calendar are joined to the schedule information, when these descriptions exist. This query contains some redundancy introduced by the application that formulated it, and makes assumptions about the nature of the data returned. Formatting of the output records into a page suitable for the Web (with only one course description presented for all four section records) was performed by an application front. Nevertheless, a considerable amount of text within the query is concerned with managing presentational issues that must be addressed by anyone wishing to make information available on the World Wide Web.

This example illustrates the utility of wrapping structured text, such as that which might be found on the Web, with relational interfaces, but it also demonstrates some of the attention to detail that is demanded by traditional database languages when dealing with missing values and in manipulating datatypes.

The query is shown on the next page, followed by two of the four records returned when the query is executed.

```
SELECT   cindex, cno, divsuf, cterm, cwt, requested, cenrolled, climit, notes, stype, sno,  senrolled,
         slimit, smt, meet_time, locn, instructor,
         COALESCE(description,'<B>'||cno||' - No Description Available</B>'),
         COALESCE(source,'')
FROM  (  SELECT  *
         FROM   ( SELECT  cindex, cno, divsuf, cterm, cwt,
                          CAST(requested AS VARCHAR(20)) AS requested,
                          CAST(enrolled AS VARCHAR(20)) AS cenrolled,
                          CAST(limit AS VARCHAR(20)) AS climit,
                          note1|| ' ' ||note2|| ' ' ||note3 AS notes
                  FROM    SCHEDULE_COURSES
                  WHERE   cno LIKE UPPER('HIST %') AND    cno LIKE '% 255%'
                  )
                  NATURAL JOIN
                  (SELECT cindex, cno, stype, sno,
                          CAST(enrolled AS VARCHAR(20)) AS senrolled,
                          CAST(limit AS VARCHAR(20)) AS slimit,
                          smt, meet_time, meet_bldg||' '||meet_room AS locn,
                          first_name ||' '||'<A HREF=/cgi-bin/nph-cgiint?__file__=calendar%
                          2Fgeneral%2Ffaculty.in&dept_name=&ftype_position=any&
                          ftype_role=none%2Fany&mode=Submit+Query&
                          back=calendar/general/schedule.in&flnm=' || last_name || '>' || last_name
                          || '</A>'   AS instructor
                  FROM    SCHEDULE_SECTIONS
                  WHERE   cno LIKE UPPER('HIST %')   AND    cno LIKE '% 255%'
                  )
         )
         NATURAL LEFT JOIN
          (SELECT CASE position('&' in TEXT_TO_STRING(cno, 'clear'))
                  WHEN 0  THEN TEXT_TO_STRING(cno, 'clear')
                  ELSE substring(TEXT_TO_STRING(cno, 'clear') from 1 for
                          position('&' in  TEXT_TO_STRING(cno, 'clear'))) ||
                           substring(TEXT_TO_STRING(cno, 'clear') from
                          position('&' in TEXT_TO_STRING(cno, 'clear'))+5)
                  END as cno,
                  TEXT_TO_STRING (KEEP_MARKS(course,0,0), 'tagged') as description,
                   '<CAL>' || TEXT_TO_STRING (source, 'clear') || '</CAL>' as source
                   FROM   (SELECT UNNEST
                                  EXTRACT_SUBTEXTS(
                                         calendar, 4,
                                          '<file>[:source#&<COURSE>#[<CNO>#]]'
                                   ) as (marked_calendar, source, course, cno)
                          FROM     uwcalendar
                          )
                  WHERE   UPPER(TEXT_TO_STRING(cno, 'insensitive'')) like UPPER('HIST%')
                  AND     TEXT_TO_STRING(cno, 'insensitive') LIKE '% 255%'
         )
ORDER BY cno, cindex, stype, sno, smt ASC
```

Result 2

RECORD 1

```
cindex:        '01131’                 cno:          'HIST 255’
divsuf:        '  '                    cterm:        'F'
cwt:           '.50'                   crequested:   '62'
cenrolled:     '53'                    climit:       '54'
notes:         '  '                    stype:        'C'
sno:           '01'                    senrolled:    '53'
slimit:        '54'                    smt:          '01'
meet_time:     '11:30TR'               locn:         'AL 124'
instructor:    'M Craton'
description:   '<Tagged><COURSE NAME="HIST255">
               <CNO>HIST 255</CNO>  <CTERM>F </CTERM>  <CWT>0.5</CWT><br>
               <CTITLE>The Expansion of England</CTITLE>
               <br>
               <CDESC> The history of the British Empire down to the American War of Independence, telling
               the story of the Tudor seadogs, of the plantation of Ireland, the settlement of the North American
               mainland, the establishment of slave plantations in the Caribbean, and the earliest British
               enterprises in Africa, Asia and the Pacific. </CDESC><br>
               </COURSE></Tagged>'
source:        '<CAL>COURSE/course-HIST.html</CAL>'
```

RECORD 2

```
cindex:        '01131’                 cno:          'HIST 255’
divsuf:        '  '                    cterm:        'F'
cwt:           '.50'                   crequested:   '62'
cenrolled:     '53'                    climit:       '54'
notes:         '  '                    stype:        'D'
sno:           '01'                    senrolled:    '19'
slimit:        '18'                    smt:          '01'
meet_time:     '12:30T'                locn:         'ES1 353'
instructor:    'M Craton'
description:   '<Tagged><COURSE NAME="HIST255">
               <CNO>HIST 255</CNO>  <CTERM>F </CTERM>  <CWT>0.5</CWT><br>
               <CTITLE>The Expansion of England</CTITLE>
               <br>
               <CDESC> The history of the British Empire down to the American War of Independence, telling
               the story of the Tudor seadogs, of the plantation of Ireland, the settlement of the North American
               mainland, the establishment of slave plantations in the Caribbean, and the earliest British
               enterprises in Africa, Asia and the Pacific. </CDESC><br>
               </COURSE></Tagged>'
source:        '<CAL>COURSE/course-HIST.html</CAL>'
```

…