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

ABSTRACT

There is a growing need 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. These tools are critical to support applications within corporate intranets and across the rapidly evolving World Wide Web.

This paper introduces a framework for modelling structured text and presents a small set of operations that may be applied against such models. 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 has been implemented to support SQL with the proposed extensions. 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. The challenge

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

Many customers require that large texts be searched both vertically, with respect to their internal structure, and horizontally, with respect to their textual content [Wei85]. Texts often need to be fragmented at appropriate structural boundaries. Sometimes selected text needs to be extracted as separate units, but often the appropriate context surrounding selected text must be recovered, and thus the selected text needs to be marked in some manner, so that it can be subsequently located within a potentially much larger context.

To support applications that manage large bodies of text, the SQL/MM standard provides interfaces for the “Full Text” abstract data type for use within SQL3 [ISO96s]. Similar approaches have been adopted by commercial systems such as Oracle Corporation’s

SQL*TextRetrieval [Ora92] and IDI's BASISplus [Sey92]. However, there is no ability to capitalize on the rich structure present in most texts nor to mark and extract subtexts as part of a Full Text search specification.

A *structured text* is 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], whereas Java source code is an example of text whose structure is determined by appropriately parsing the language contained within the text [Fla96]. Elsewhere such data has been termed "semi-structured" to distinguish it from rigidly structured business data as found in relational databases [Abi97a, Suc97].

Recognizing that text retrieval is an important component of effective information systems, Fulcrum Technologies Inc., Grafnetix Systems Inc., InContext Corporation (now EveryWare Development Inc.), Megalith Technologies Inc., Open Text Corporation, Public Sector Systems, SoftQuad Inc., and the University of Waterloo formed the Canadian Strategic Software Consortium (CSSC) in 1993. CSSC's goal was to pursue pre-competitive research relating to the integration of relational databases and text-intensive databases [CSSC94]. Commercial realities dictated that we explore how relational database systems could be extended, so that they could effectively provide access to structured text in a manner compatible with SQL. Early versions of our proposals have been presented with examples [Bla94, Bla95].

Others have also proposed data models and languages that could be adopted to manage structured text databases. HyQ[ISO97] and DSSSL [ISO96d] are two ISO standards developed to query and manipulate structured text, but neither is integrated with a database language. The Lore project and the Lorel query language are based on modelling semistructured data as a rooted, directed, labelled graph [Abi97b], and a similar model underlies the UnQL language [Bun96]. The database implementations underlying both projects are based on storing materialized versions of the graphs. Unlike these approaches, we insist that the text remain intact as the authoritative repository of text data, and a graph-like *view* be defined over the text. The object-oriented language O_2SQL was extended to support SGML[Chr94] as was OSQL[Yan94], and many ideas in this paper have evolved from similar considerations; one major extension we provide is to maintain information about the context of selected text as well as extracting the subtext. Closer to the relational model, Atlas was developed as a nested relational database system, in order to maintain structured text in a single relational field, and an extended SQL language was defined to provide text support [Sac95]. Our approach, unlike these others, is to design operators that are naturally and efficiently supported by *off-the-shelf* text search engines and easily integrated with *standard* SQL systems.

If text is to be embedded within conventional databases, we must be able to define views over text that are accessible from those systems, without destroying the texts themselves and without undesired duplication of data. What is initially required is a simple text framework that is compatible with the rest of the database system's model and encapsulates most of the significant properties of structured text. The database query language must then be coupled with well-defined operations on texts that facilitate effective query, retrieval and update of selected text fragments [Ray96a].

As well as pursuing research in language design and support, we were also interested in 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]. The resulting system is similar in intent to TSIMMIS [Gar97] and other mediator-based systems, except that the component interfaces are strictly ODBC [Mic92].

In this paper we describe the final text modelling framework that we developed to meet the varied uses of structured text in a database environment. The semantics of our proposed language have been formally defined [Dav96], and the extensions have been adopted for inclusion within the SQL/MM standard [ISO96s]. Our proposed definition of the structured text abstract data type is presented in Section 2. Section 3 describes a set of related applications that illustrate the utility of such structured text objects. A brief critique of the ADT and suggested extensions to explore are given in Section 4, and conclusions follow in Section 5.

2. Structured text objects

In this section, we describe support for tree models for structured text, including operations on such models that provide the functionality needed to select and extract subtexts in an object-relational database environment. Because the text ADT is intended to be used within a more general database environment, standard object-relational operators that manage lists, sets, and tables are assumed to be defined, and these complement the operators on text. Furthermore, access to related data that instantiate datatypes other than text (or that result from converting small text fragments to other types) is provided by other system components. Therefore the principal functionality to be supported within the ADT is to locate and extract desired subtexts to form collections that can be further manipulated by the database system.

The specific operators we developed for the structured text ADT were designed to be compatible with text search engines, SQL3, and SQL/MM, and they have been influenced by SQL's syntax and semantics. Nevertheless, because the resulting ADT was designed to meet the demands of our industrial partners familiar with commercial applications involving structured text, it forms a useful type definition for any object-oriented or object-relational database environment.

2.1. Tree models of structured text

A structured text subsumes a region of text that 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 [ISO86], 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 structure that the node represents conceptually, and each node contains as a second attribute the subtext subsumed by this structure.

To interoperate with texts, we introduce the first three functions of the text ADT:

Function	*Arguments*	*Returns*	*Description*
string_to_text	String , Method	Text	Parses a string to form a text
text_to_string	Text , Method	String	Forms a string from a text
cast	Text	*native type*	Casts a text as integer/double/date etc.

Figure 1. Text creation and interpretation

The function **string_to_text** takes as input two arguments. The first is a string containing the sequence of characters to be parsed, and the second is a keyword (passed as a string in SQL) 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. Two texts that are parsed using identical arguments (*i.e.*, equal input strings and identical methods) are said to share the same *provenance*.

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 might produce a tagged string, a second might omit all tags, and a third might suppress particular subtexts.

Suitably encoded texts (*cf*. [Gon87]) can be directly **cast** into numeric integers, double precision values and dates, without first being transformed into strings. This allows large relations to be encoded within a text directly while continuing to be rapidly accessible.

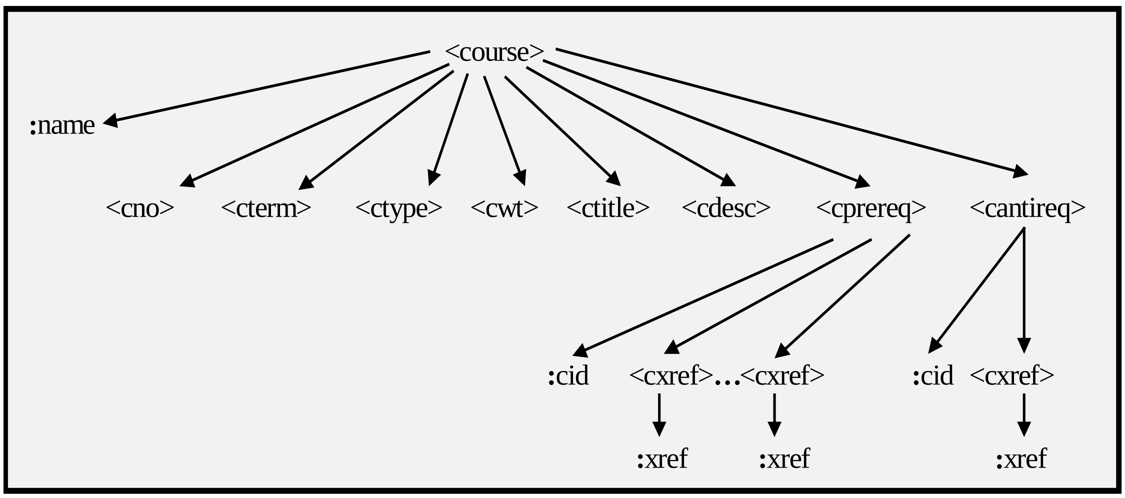
Consider the fragment of the University of Waterloo calendar shown in Figure 2 [UW98]. This fragment could be encoded as a tagged string (Figure 3a) and interpreted by a parser as a labelled tree (Figure 3b), where the text values subsumed by each node within the tree are not shown. The text corresponding to Figure 3b may have been produced by applying the **string_to_text** function to a string formed from a document type description (DTD) appropriate for the calendar and the contexts of Figure 3a, passing the keyword "SGML" as the second parameter. This text is not necessarily materialized, but rather represents a view of the character string against which searches can be applied.

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 2. Part of Chapter 16 of the University of Waterloo calendar

```
<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 3. An encoding for a fragment of the calendar

In this sample model, labels in the tree are “typed” as being SGML generic identifiers [ISO86] by the convention of using enclosing angle brackets**,** whereas attribute names are preceded by a colon. A node representing a generic identifier subsumes the subtext within the corresponding tags, and a node representing an attribute name subsumes 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). Furthermore, in this example some SGML markup has been ignored by the modeller, as has the case used in the string for generic identifiers and attribute names. Other SGML types (such as entity references) may be similarly encoded.

Continuing with the 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. (Mechanisms to search for ordered and unordered text segments are described in the next section.) However, 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.

This framework allows the full generality of structured text models to be exploited by diverse applications. Out of necessity, we have chosen one particular realization of the framework for this example. In general, individual *enterprise designers* determine the structures that are to be identified in the conceptual model and how type information is to be encoded within node labels. No assumptions are made in the framework about what constitutes structural information within an arbitrary text; this is imposed 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 stored, since such assumptions would limit the usefulness of the model [Mac92]. *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], enterprise and applications designers 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.

An 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 framework 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 framework. 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, Spe94, Ray96b]) or to texts that correspond to arbitrary directed graphs (*cf*. [Abi97b, Bun96]). We have found in practice, however, that tree models are generally flexible enough for most commercial applications.

2.2. Marking structured text

Previous proposals (such as [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 identified within our framework, any node within a text tree may be either marked or unmarked. Thus, an instance of structured text not only identifies spans of subtexts, but also includes a set of zero or more marks that identify selected structured subtexts.

The functions in Figure 4 allow marks within a structured text to be manipulated. Where multiple texts are provided as parameters, they must share the same provenance (*cf*. union *compatibility* in SQL); otherwise an appropriate exception is raised. For all six functions, the text returned shares the same provenance as the text parameters.

Function	*Arguments*	*Returns*	*Description*
mark_subtexts	Text, Pattern	Text	Mark matched subtexts
union_marks	Text, Text	Text	Combine marks from two instances
intersect_marks	Text, Text	Text	Intersect marks from two instances
except_marks	Text, Text	Text	Subtract marks from two instances
keep_marks	Text, Int, Int	Text	Preserve subsequence of marks
aggregate_marks	TextSet	Text	Union marks over set of instances

Figure 4. 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 was designed to be compact, yet expressive, and to be compatible with full-text searching as defined for SQL [ISO96s]. The pattern language is illustrated in Figure 5, assuming the text model implied by Figure 3. (Following SQL's conventions, '%' matches zero or more consecutive characters within a text label. The hash mark '#' identifies which nodes are to be marked upon a successful match.) The language is formally defined using the BNF for <*pattern*> in Figure 6. An alternative syntax using more descriptive function names rather than the compact notation presented here has also been defined [Dav96].

Pattern	**Marks**
<cref>#	Every <cref> node (cross-reference) in the text
<cref>[^%#]	Every child of any <cref> node in the text
@<course>#[<cprereq>]	Every marked course having a <cprereq> child
<course>#[<cprereq>[:xref{CS370}]]	Every course that lists CS370 as a prerequisite
<c%req>#[:xref{CS230}&:xref{CS246}]	Relations to both CS230 and CS246 in either order
@<course>#[<cwt>,<cterm>]	Marked courses whose weight appears before term
<course>[:name{CS370}&<cwt>#]	The course weight of CS370
%[<cno>#,<ctitle>#]	Every pairing of <cno> followed by <ctitle>
^%#	The root of the text (regardless of label)

Figure 5. Examples of how structured text patterns match a text

```
<pattern>           ::= <node_rule> [ <descendants> ] | <node_rule>
<descendants>       ::= <set> | <list>
<set>               ::= <pattern> & <set> | <pattern>
<list>              ::= <pattern> , <list> | <pattern>

<node_rule>         ::= <rooted_rule>
<rooted_rule>       ::= ^ <marked_rule> | <marked_rule>
<marked_rule>       ::= @ <marking_rule> | <marking_rule>
<marking_rule>      ::= <node_pattern> # | <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 !!
```

Figure 6. The structured text pattern matching language

The *<pattern>* and *<descendants>* productions, 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; 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 symbol '**^**' matches a node whose parent node (if any) is also simultaneously matched by its corresponding *<node rule>*,
(e) each *<node rule>* containing the symbol '`@`' matches a marked node within the input text,
(f) every *<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 function **mark_subtexts** identifies all possible matches (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 hash symbol '**#**'.

The rules governing how node labels and subsumed text are matched against strings within the pattern tree should be compatible with the environment within which the structured text abstract type is supported. For SQL, a *<node label>* may use the symbols '`%`' and '_' as wildcards, such a *<node label>* is compared with structured text labels using the SQL 'like' predicate [ISO92], and this comparison is case insensitive. Furthermore, the *<text_expression>* must 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, a <*text expression*> could 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 the detection of 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]`). For example, every course that lists CS370 as a pre-requisite could be marked by the pattern `<course>#.<cprereq>..:xref{CS370}`. Convenient notation to match descendants (as oppose to direct children only) reflects the ability of text search engines to find contained strings efficiently.

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, or 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 (starting from 1) consistent with the order that they would be visited by a pre-order traversal of the text tree; those marks within the input text having ordinals lying in the specified range are preserved in the resulting text. This function provides a simple mechanism to identify the subtexts based on their order in the text (*e.g.*, the second through fourth figures).

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. This function may be applied as an aggregation function in SQL, for example, after grouping related rows in a table.

Two other related functions are also defined to simplify the coding of applications, one to test for matching text without marking and one to count marks in a text:

Function	*Arguments*	*Returns*	*Description*
text_match	Text, Pattern	Boolean	Matches text against a hierarchical tree pattern
count_marks	Text	Integer	Counts the number of marks in a text

Figure 7. Other functions associated with structured text

Having marked a set of nodes in a text, these can be referenced in subsequent text patterns through the flag **@** to focus further matching. In the next section, we show how text marked as a result of pattern matching and suitable combining of marks can be extracted into tables for further manipulation as atomic entities outside the text ADT.

2.3. Extracting structured subtext

Both functions shown in Figure 8 extract from an input text a collection of subtexts, returning a relation that contains the extracted subtexts.

Function	Arguments	Returns	Description
isolate_subtexts	Text	Relation	Extracts all marked subtexts within a text
extract_subtexts	Text, Int, Pattern	Relation	Extracts subtexts matching a given pattern

Figure 8. Functions that extract subtexts from a text

The function **isolate_subtexts** creates a binary relation. It takes an instance of text as input and, for each mark within this text, produces an output row. The first attribute within each output row contains 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.

This form of output preserves the context of a match together with the isolated text on its own. The pairing in a single row maintains the relationship between these two texts in a manner that suits SQL's value-based semantics. Subsequent operations in SQL can select and project data from this relation to suit various applications' needs.

The function **extract_subtexts** takes as input an instance of text, an integer, and a structured text pattern as described for **mark_subtexts** above. It produces a multi-column relation with one row for every possible complete match, as described below. The number of columns in the resulting relation depends in principle on the text pattern, but the string containing the text pattern can be constructed dynamically and the number not known until runtime. In environments where this value must be known at compile time, the middle parameter indicating the expected number of columns in the resulting relation must be included. In SQL2, for example, this middle 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.

In the pattern, let the number of <*node rule*>s flagged with a hash mark be *n*. For every distinct matching of the *n* flagged <*node rule*>s within the structured text pattern against nodes in the structured text (matching the entire structured text pattern against the text in at least one way) an output row is produced with *n+1* columns. The first column contains a text with the same provenance as the input text, while the remaining *n* columns contain the extracted 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 3 the relational rows shown in Figure 9a are returned in no specific order. (Within this figure marked subtexts within a text are printed in boldface and underlined.) This relation differs substantially from that returned by the corresponding use of **isolate_subtexts** as shown in Figure 9b:

```
isolate_subtexts(
  mark_subtexts(calendar,’<course>[:name#, <cxref>..:xref#]’)
  )
```

<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”

(a) Result returned by **extract_subtexts**

<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”

(b) Result returned by **isolate_subtexts** on a text marked using the same pattern

Figure 9. Comparison of text extraction operators

Applications that rely on correlated subtexts will require the approach based on **extract_subtexts**, whereas the alternative is simpler to use when one part of an application identifies interesting subtexts and another one manipulates them. In both cases, the ability to extract a subtext while preserving the context within which it was extracted is significant to meet the varied needs of diverse applications. Furthermore, it is important to note that all elements of the relations returned are of type *text* and each can be represented as an unmaterialized view over the originally parsed character string.

2.4. Associating a schema with text

Individuals and computer processes that access a data source may initially have little knowledge about the information contained within this data. It is therefore important that all data sources be self-descriptive.

The SQL2 standard requires the presence of an information schema, having a universally understood data structure that contains meta-data providing detailed information about the structure and content of each SQL2 database catalog. Because this information is provided in a relational format it can be retrieved using appropriate SQL commands. Using information recovered from the SQL2 information schema, it is possible (subject to appropriate permissions being granted) to formulate, validate and initiate SQL2 operations that manipulate or return information from any part of the described SQL2 database catalog.

While it might be naively assumed that the structural schema associated with a text can be deduced from an examination of the text directly, this is not true. Specific structural elements may not be present within the text retrieved, and the structural layout for a fragment of text would contain no reference to absent structural components. Furthermore, one instance's structure contains no record of the specific constraints imposed by an underlying grammar.

The encoding shown in Figure 3b provides a conceptual model of the structure of the text shown in Figure 1, but it fails to provide key information needed by a user who wishes to access such text more generally. Such a user, and more importantly an application program, 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 3 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 for parsing.

The schema shown in Figure 10 describes the actual information content present within Chapter 16 of the University of Waterloo calendar, and constitutes part of the schema for the calendar. This diagrammatic form is a visualization of the grammar, showing, for example, that a `<file>` element contains an attribute `:source` and one or more subelements having identifier `<cdept>`. (Repeating elements within this schema have been marked with a '+'.) In summary, 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.

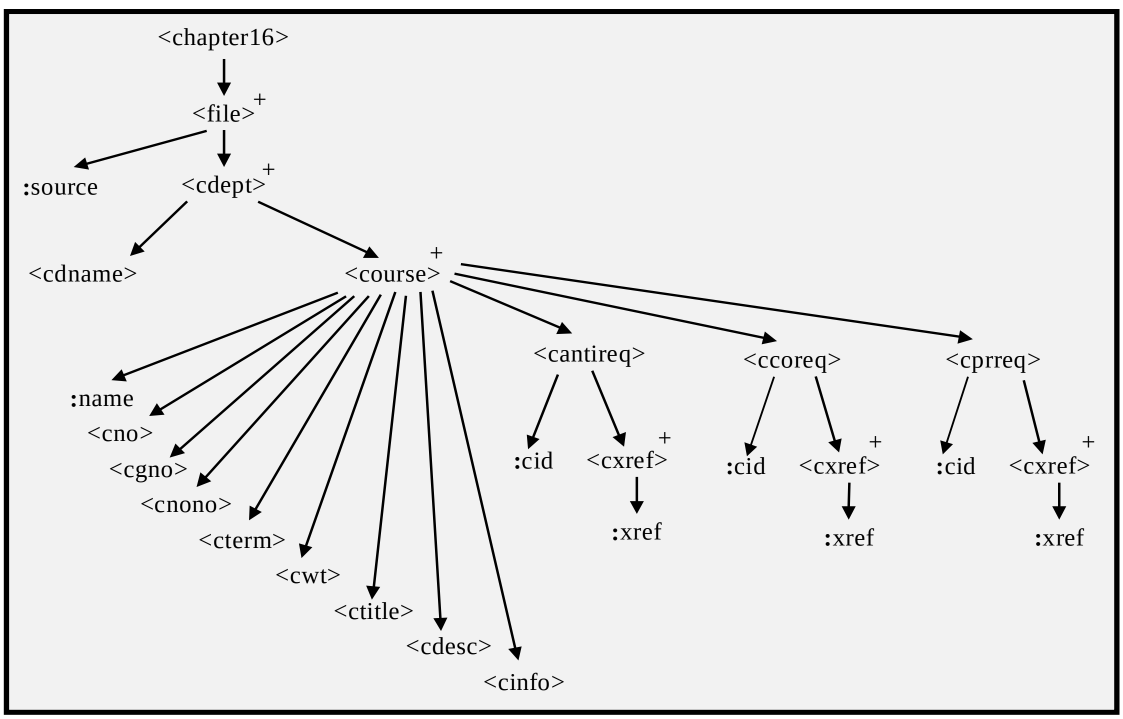


Figure 10. A schema for Chapter 16

If such structural schemas were not available, users and applications would have little ability to discern the hierarchy of structure contained within accessed texts and to associate appropriate labels with structural components. As a consequence they will be unable to formulate, validate, and initiate structured text operations against the data.

One can consider a grammar ADT to be a specialization of the structured text ADT. Thus, if the schema is itself considered as structured text, the operators defined earlier in this section can be applied to manipulate it. Furthermore, certain additional operators are defined to simplify application code written to query a text's grammar. Specifically, the functions shown in Figure 11 provide rudimentary access to the grammar associated with a text and enable programmatic access to the schematic information presented above.

Function	*Arguments*	*Returns*	*Description*
text_to_grammar	Text	Grammar	Returns the grammar for a given text
grammar_root	Grammar	String	Extracts the name of the root of a grammar
grammar_elements	Grammar	Relation	Returns element names with their descriptions
grammar_hierarchy	Grammar	Relation	Returns child/descendant relationships
grammar_to_text	Grammar	Text	Full textual description of a text's grammar

Figure 11. Functions on the schema of a text

The function **text_to_grammar** returns a normalized grammar associated with a given text. (Note that although all examples here use SGML, this is not a requirement imposed by the model. However, in our implementation, the normalized grammar is automatically extracted from a DTD when an SGML parser is applied to create a text through the function **string_to_text**.)

The function **grammar_root**, when applied to such a grammar, returns the label of the root of the schema for this grammar; this is a special case of **text_to_string** as applied to a structured text that happens to represent a grammar. The function **grammar_elements** (built on **mark_subtexts** and **isolate_subtexts** invocations that are tailored for a grammar) returns a binary 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 12 when applied to the calendar schema; the descriptive information forms part of the information to be provided to the parser (for example, as comments in a DTD).

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

Figure 12. 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 can be defined in terms of **extract_subtexts**, returning the table shown in Figure 13 when applied to the calendar schema.

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

Figure 13. Part of the relation returned by **grammar_hierarchy**

As stated earlier, queries involving arbitrary descendants are more efficiently processed by text engines than those involving direct descendants only. By including all descendants explicitly in the grammar hierarchy, an application can determine which paths are legal in text patterns without first having to call a potentially expensive transitive closure function (which is also non-standard in SQL). For applications that desire direct descendants only, a simple selection of tuples having `Relationship='Child'` produces the desired result. Using the data in a table such as that shown in Figure 13, an application can build interactive query interfaces for users or it can use the data directly to build meaning pattern strings to be used to query and extract data from a text.

The **grammar_to_text** function is not the inverse of the **text_to_grammar** function, but instead produces a structured text corresponding to a parsed string-form of the grammar. For example, the text of the grammar associated with an SGML document corresponds to the original document type definition (DTD) in its string form. If no such text exists, the function returns null. 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, or a particular abstract grammar, but against any textual description of the grammar associated with a text.

It is important to note that the framework 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 text objects. Based on industrial practice, it is unrealistic to demand that only texts having exactly the same grammatical schema can be grouped into collections. Even in applications based exclusively on SGML, business needs and practices evolve at a pace through which DTDs are modified almost as rapidly as individual texts, and far more frequently than conventional database schemas. Grammars obtained from an instance of text satisfy value-based semantics, and they can therefore be independently compared, manipulated, stored, and retrieved. Thus, in applications where tighter control need be exercised on the forms of related texts, database designers may choose to impose constraints on text collections (*e.g.,* all texts in one column) to ensure that the grammars of included texts share certain features. Furthermore, grammars can also be independently used to document and constrain texts even before such texts are stored within a database.

3. A sample application

We have developed and tested the ideas presented in Section 2 in the context of a federated database environment created as part of the CSSC project [CSSC94, Bla95, Bri97]. In brief, the implementation uses an architecture based on independently managed data sources, *wrappers* that provide common interfaces to those sources, and a *mediator* that serves as a hybrid query processor providing access to the data sources through their wrappers and integrating the results (*cf.* TSIMMIS [Gar97]).

Wrappers have been written to translate the proposed text operations into native language constructs supported by Open Text's PAT 5.0 text engine [OTC95] and to those supported by the MultiText text engine [Cla94]. Similar wrappers could be written to interact appropriately with object-oriented database systems and with other structured text search engines, using alternative native search languages.

The wrappers provide an SQL2 interface to the hybrid query processor, using ODBC for communications [Mic92]. Within the mediator, subtexts output from these operations are integrated with texts stored in a Fulcrum database [Ful94] and with relational data stored in Oracle and DB2 databases. The resulting federated database system provides efficient access to structured text and to relational data more generally [Ng97, UW97]. Extended explanations and demonstrations can be viewed through the Web at http://solo.uwaterloo.ca/trdbms/. In the remainder of this section, we illustrate the ideas using one of the demonstration databases.

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 [Ben98].

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 browsing through all relevant sections of the calendar, 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], while maintaining its text intact. Thus, retrieved information can be read in its original written form, so that the database does not obscure the intended semantics of the calendar's contents (*cf.* [Tom89]).

After automatically replacing the HTML tags in the calendar with more descriptive SGML markup (as shown in Figure 3a), the resulting document was indexed so that it could be rapidly searched by Open Text's 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 text managed as 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. Furthermore, the DTD describing the descriptive structure associated with the various sections within the calendar formalized the implicit rules governing how various departments prepare material for inclusion within the calendar. As a result, the University moved closer to standardizing and automating the data entry process associated with construction of a yearly calendar. These benefits, however, accrue directly from adopting SGML for representing text [Coo87].

Having added descriptive markup to the text, database management made it possible to validate textual information contained within the calendar more easily. A text view corresponding to the complete calendar was created as a one-row, one-column relation accessible through a wrapper interface to Open Text's engine. Using the text operators presented in Section 2, it was then easy 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 [UW97]. This information can be validated against corresponding information in a current telephone directory stored within an Oracle database. Alternatively, relational information derived from the calendar can serve as the source data to be exported directly to conventional relational database systems for use in alternative applications.

Making the calendar's text available as a database made it 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. The details of one such query are given in Appendix A (Query 1). Of course, such queries can also be supported directly by structured text engines. However, it is possible to perform very much more complex queries using the expressive power inherent in SQL2 in conjunction with the structured text ADT described here.

For example, 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 extended with the text operators to derive a table from the calendar that documented the number of members of faculty at various ranks by department; this would not be possible through a conventional text search engine. Using conventional SQL, 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 an Oracle database. The hybrid query processor could then integrate the collective information into the desired relational tables, by joining the relations from the distributed data sources.

As a second example, we show in more detail how the text ADT has improved access to the University of Waterloo calendar. A Waterloo student was interested in courses relating to Ireland. Through a forms interface to the calendar database, he found that History 255 "The Expansion of England" was the only course within the calendar to include the word Ireland within its course description. With another click, he was then able to recover the course schedule associated with this course (see Query 2 in Appendix A). 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.

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, an unmaterialized relational view `course_associations`, capturing all documented course pairings, was easily defined using the extended text operators against the text of the calendar. A form providing access to this view was quickly added to our demonstration and made available for use by members of the Registrar's office and others (Figure 15).

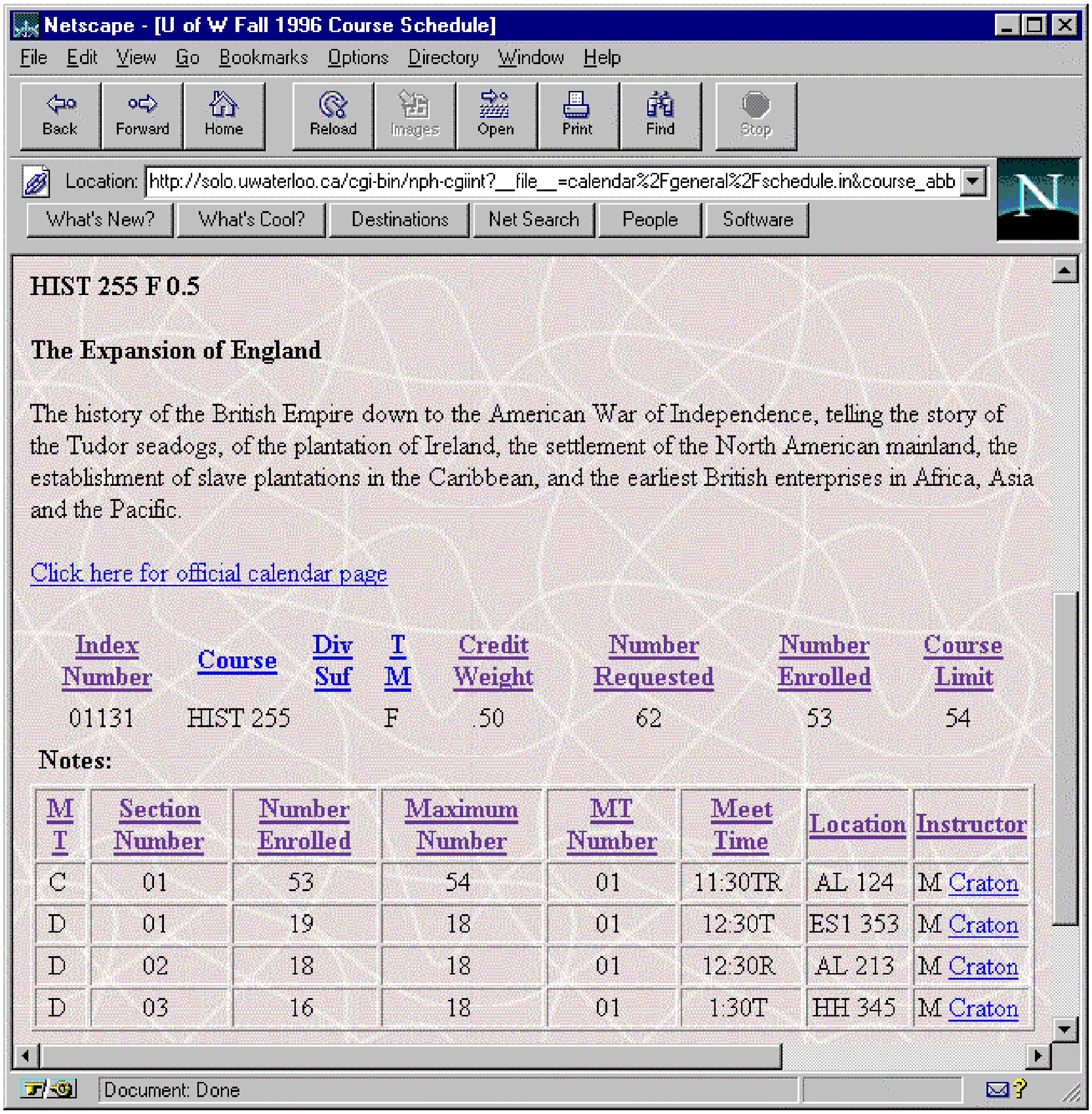


Figure 14. Output that relates structured text with relational data

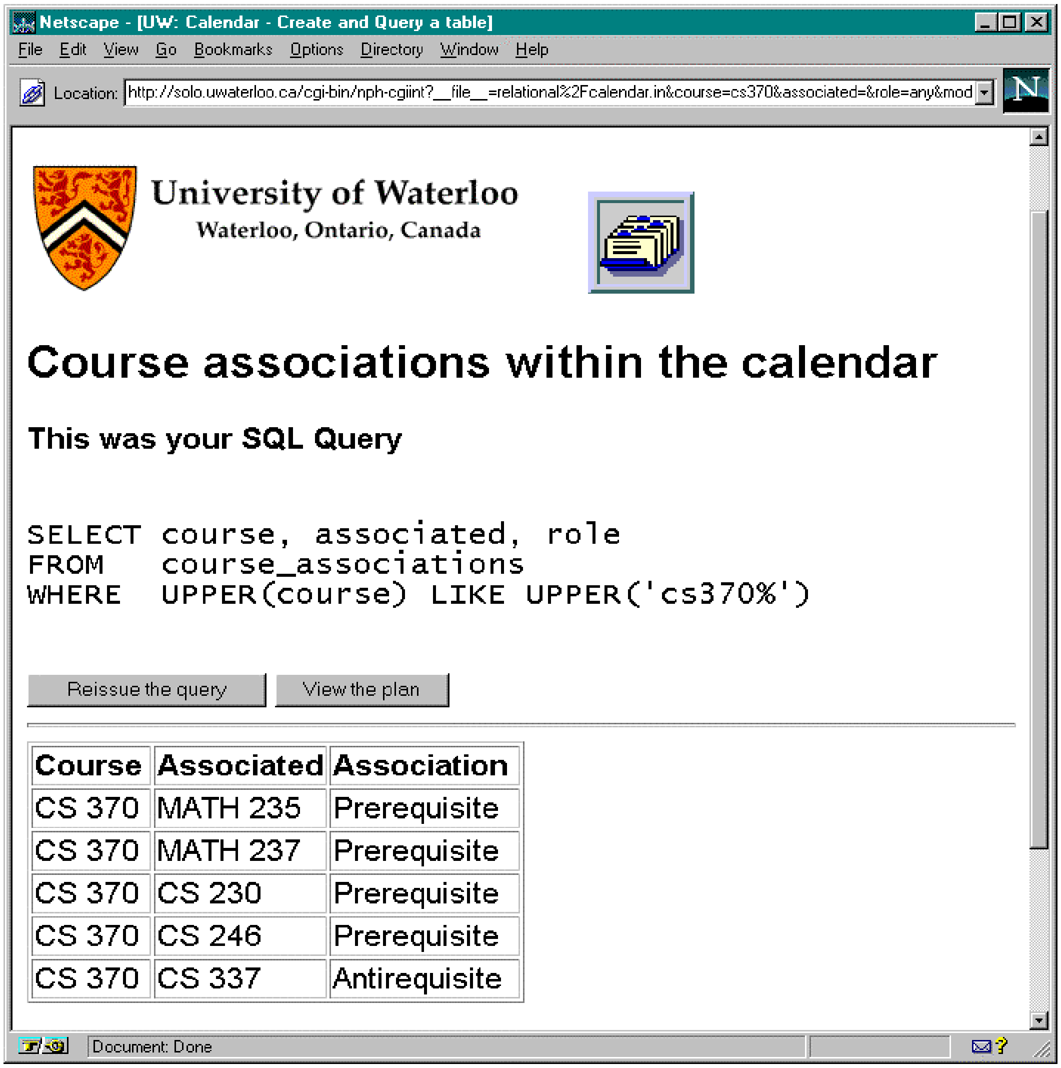


Figure 15. Presenting course associations as a relation

The Student Awards office asked us to provide access to financial award information contained within the calendar so that end users would be able to search for awards, grants and scholarships, using various criteria, including numeric considerations associated with an award. Having identified those fragments of text that describe applicable awards, 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 advantages of accessing text through SQL over other approaches is evident here: loading a database with summaries of the data *in lieu of the text itself* will necessarily omit important descriptive information, and storing this data redundantly is wasteful and error-prone; however, complex numeric processing and aggregation is not supported by conventional text search engines. Full SQL access to text is obviously an improvement.

4. Critique of the Text ADT

The structured text ADT presented in Section 3 has been useful in supporting several diverse applications. In fact, the SQL extensions for structured text have been accepted for inclusion within the evolving SQL/MM standard [Dav96]. In spite of this success, however, some extensions and variations of the ADT might be usefully defined.

At present SQL/MM's 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 text [ISO96s]. These concepts should be viewed as specific instances of well-defined structure associated with a structured text being searched. Ideally, full interplay should be allowed between "horizontal" (Full Text) and "hierarchical" (structured text) searching and marking, thus making the resulting language much more expressive. To accomplish this, the SQL/MM Full Text specification must be augmented to allow marking of identified substrings matching Full Text patterns. On the other side, the structured text ADT should be augmented to allow the concept of proximity, which is well defined within the Full Text proposal, to be applied to structured text.

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). Operators to handle hypertexts may be found to be useful additions to the proposed ADT. If such operators require that the data models be extended to arbitrary directed graphs, it may be useful to interpret the graphs' labels as being on edges rather than nodes. Although labelling nodes or edges is equivalent for trees, this change would allow labels to reflect the various roles played by the data in a more general graph [Abi97b].

Structured subtexts may contain or reference heterogeneous objects such as image, sound, video, spatial and other data types frequently found within existing multimedia documents, and these objects may themselves contain or reference further instances of structured text. The tree-structured model on which the proposed ADT is based can easily accommodate multimedia objects in place of text at its nodes. Selection of these objects based on node labels, matching objects' contents using braced patterns appropriate to the objects' types, and extraction of the objects into relations is still applicable (with appropriate calls to **cast** to realize the objects in their native types). Structured multimedia objects may also exist as temporal objects, having differing manifestations at different times, all of which must be accessible by specifying the appropriate temporal context. Again, suitably labelled trees (or graphs) can model temporal variants, but we have not yet had experience with such general hypermedia environments.

The proposed framework is well-suited to text that is physically present but considered within some contexts to be logically absent (*e.g.*, colour, white space, punctuation, descriptive markup, versioning): some pieces can be ignored by the parser when the text model is built, and other pieces can be ignored by the application user when subtexts are extracted. However, certain text may be considered to be logically present while being physically absent (*e.g.*, SGML attribute values and C++ function parameters may be assigned default values when absent). It is recommended that the parsers used to convert strings to text build models that include default values in their text trees so that they can be queried as if explicitly present. In practice, however, this implies that the texts themselves be altered to include those values prior to being indexed by search engines or that indexing technology that includes virtual terms be adopted by the engines.

One problem inherent in structured text databases is irregularity in the structure [Abi97a]. Therefore, applications often need to specify a structural pattern in which some of the subtexts may be absent; it is desired that such optional components be marked if present but their absence should be ignored. Using our operators, 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. For example, finding all courses that list CS370 as a prerequisite and marking those courses *and their co-requisites if present* requires a two-part query: optional subtexts must be marked and extracted in a second phase, after mandatory subtexts have been extracted. This second extraction phase is 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.

To make matters worse, the extracted subtexts corresponding to optional components cannot be easily related to their contexts. In addition, the separation 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. In our applications, the mandatory and optional subtexts are recombined through the use of an appropriately constructed outer join, and absent subtexts are represented within such an extraction process by null. Extensions to the proposed pattern matching language that would provide direct support for optional matching of text should be considered (*cf*. optional matching in the context of specific semistructured data in OEM-OQL [Abi97b]).

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 rank how well instances of subtexts match a given search specification, and to recover (in a suitable order) those subtexts that exceed some specified ranking threshold. Such a facility would also address the problem of optional subtext matching, since optional components could be assigned a relatively small weight within the overall ranking scheme. This direction needs further exploration.

Associating an appropriate grammatical structure with instances of extracted subtext is difficult when the context from which these text has been extracted is lost, as is the case in value-based semantics. Which of the schematic constraints still apply to extracted subtexts? For example, if three paragraphs are extracted from a section that forbids the inclusion of footnotes, perhaps the extracted paragraphs should no longer maintain the exclusion. However, if in the original context the paragraphs were considered to be ordered, this ordering constraint should likely still be enforced. Even maintaining the subtext's tree model (as shown in Figure 3) might not be straightforward: if a page break occurs between two of the three paragraphs in the original context, is this page break also present within the resulting structure associated with the three *extracted* paragraphs? Because the proposed ADT allows the relationship between a text and those subtexts extracted from it to be preserved, it is possible to delegate application-specific requirements pertaining to extracted subtexts to the application domain, where they can be appropriately addressed.

Unfortunately, since the context is preserved in the *containing* text (by marking those subtexts extracted from this text), it becomes difficult to identify precise context when multiple concurrent extractions are performed against a single instance of text. This is because it may be 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 9a, 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 <*marking rule*> production shown in Figure 6 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. We have not yet had experience with this operator, but we believe it will prove to be a useful addition to the proposed standard.

Finally, the operators defined for the *grammar* subtype, while certainly useful, are by no means complete. 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 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 which relationships are one-to-one and which are one-to-many.

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 extensions for structured text are of immediate benefit to users who wish to integrate textual information into their existing relational database systems, and to users currently involved in text intensive searching or querying who wish 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 (now EveryWare Development Inc.), Megalith Technologies Inc., Open Text Corporation, Public Sector Systems, and SoftQuad Inc. CSSC 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 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

[Abi97a] S. Abiteboul, "Querying Semistructured Data," *Proc. 6th Int. Conf. On Database Theory (ICDT'97)*, Delphi, Greece (January 1997), *Lecture Notes in Computer Science 1186*, Springer-Verlag, 1-18.

[Abi97b] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J.Weiner, "The Lorel Query Language for Semistructured Data," *Journal of Digital Libraries 1*, 1 (April 1997) pp. 68-88.

[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.

[Ben98] Bender, B.L. (ed.), *Undergraduate Studies Calendar 1998-99*, Undergraduate Recruitment and Publications, University of Waterloo, (http://www.adm.uwaterloo.ca/infoucal).

[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.

[Bun96] P. Buneman, S.B. Davidson, G.G. Hillebrand, D. Suciu, "A Query Language and Optimization Techniques for Unstructured Data," *Proc. 1996 ACM Sigmod Int. Conf. on Management of Data* (Sigmod 96), Montreal, Canada (June 1996), *Sigmod Record 25*,2, pp. 505-516.

[Chr94] V. Christophides, S. Abiteboul, S. Cluet, M. Scholl:, "From Structured Documents to Novel Query Facilities," *Proc.1994 ACM Sigmod Int. Conf. on Management of Data* (Sigmod 94), Minneapolis (May 1994), *Sigmod Record 23*,2, pp. 313-324.

[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.

[Gar97] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. "The TSIMMIS Approach to Mediation: Data Models and Languages," *Journal of Intelligent Information Systems 8*,2 (March 1997) pp. 117-132.

[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.

[ISO97] International Organization for Standardization, *Hypermedia/Time-based Structuring Language (HyTime) - 2d Edition*. ISO/IEC 10744:1997.

[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

[Ng97] K.-Y. Ng, *The Use of a Combined Text/Relational Database System to Support Document Management*, University of Waterloo Department of Computer Science Technical Report CS-96-07 (January 1996) 121 pp.

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

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

[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).

[Sac95] R. Sacks-Davis, A. Kent, K. Ramamohanarao, J.A. Thom, , and J. Zobel, "Atlas: A Nested Relational Database System for Text Applications.," *Trans. Knowledge and Data Engineering 7*,3 (1995) pp. 454-470.

[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.

[Sey92] Seybold Publications, "IDI Pursues Document Management," *Report on Publishing Systems 21*,16 (May 1992).

[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/).

[Suc97] D. Sucio (ed.), "Special Section On Management Of Semi-Structured Data," *Sigmod Record 26*,4 (December 1997).

[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.

[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.

[Yan94] T.W. Yan, J. Annevelink, "Integrating a Structured-Text Retrieval System with an Object-Oriented Database System," *Proc. 20th Int. Conf.on Very Large Data Bases (VLDB '94)*, Santiago de Chile (September 1994), Morgan Kaufmann, pp. 740-749.

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

Appendix A

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 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 SQL2, which allows projected functions that return relational tables to be unnested [Bla95]. Within SQL3 it has been proposed that such an operation 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 subtexts for 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 frontend. 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> <CTITLE>The Expansion of England</CTITLE> <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> </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> <CTITLE>The Expansion of England</CTITLE> <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> </COURSE></Tagged>′		
source:	′<CAL>COURSE/course-HIST.html</CAL>′		

...