# Toward the Union of Databases and Document Management: The Design of DocBase[†]

Arijit Sengupta
Department of Computer Science
Lindley Hall 215
Bloomington, IN 47405
+1 (812) 855-3703 (voice)
+1 (812) 855-4829 (fax)
asengupt@cs.indiana.edu (email)

**Abstract**

With the advent of the World Wide Web (WWW) and the increased use of electronic documents in almost all aspects of computing, the problems of management of and systematic information retrieval from electronic documents have become highly pertinent. Information retrieval (IR) techniques allow us to retrieve documents based on keywords, but often these searches are not powerful enough to accurately extract the most relevant information. Most IR systems are designed for broadening the scope of search. However, extracting only documents highly related to the search is often desirable to keep the result set small. We propose a method for achieving very powerful searches on tagged documents by using the structural information in the tags as meta-data in queries. We adopt SGML as our tagging format. Since HTML is an application of SGML, and XML is designed as a subset of the SGML standard, our work is immediately applicable to the current and future incarnations of the WWW. In this paper, we give an overview of the methodologies (such as query languages, query interfaces, and query processing techniques) used in the design of *DocBase*, our prototype proof-of-concept document database system. DocBase is a modular system capable of performing SQL-like queries on native SGML documents using pluggable indexing and storage-management applications. Because of the generalized nature of SGML, and the ever increasing use of structured documents in the corporate world, we argue that systems like this will be indispensable in the forthcoming century.

**Keywords**: SGML, XML, SQL, query languages, query processing, document databases, structured documents, information retrieval

## 1 Motivation

With the growth of the WWW and the dramatic increase in the number of electronic documents in use, the problem of extracting relevant information from these documents have become very pertinent. A common frustration of the users of the WWW is the enormity of search results, few of which are actually highly relevant to the search problem. Most of the current search engines on

[†]This work was partially supported by US Dept. of Education award P2000A502367 and NSF Research and Infrastructure Grant, NSF CDA-9303189

the WWW are based on keyword-based searches. Obtaining relevant search results is difficult even with multiple boolean combinations of keywords. Our solution is to utilize the embedded structural information in the documents by involving meta-data in queries. For instance, it would be simple to restrict the searches in particular regions of the documents denoted by the tags that mark the structure in the document. We argue that the increasing use of HTML (and in the near future, XML) will make the use of this approach quite feasible.

HTML, the markup language of the WWW, incorporates some structural information in documents in the form of HTML tags. However, since most of the HTML tags are based on layout, they do not convey useful structural information. Fortunately, a considerable number of documents are published in SGML, (Standard Generalized Markup Language)[ISO86]. In SGML, the structure of the document is defined using a DTD (document type definition) and documents are defined as **valid** instances of the DTD. The upcoming XML standard [W3C97] is based on SGML and has the potential to be the format for the WWW of the future.

The objective of this work is to develop a methodology for processing SQL-like queries on structured documents *in place*, without requiring to map the documents into different formats or to import them into a different system. This will allow users to search the document repository using queries that are powerful and can accurately extract document fragments based on the given criteria. SQL constructs such as selections, projections, joins, nesting, grouping and ordering give the much needed power to the searches on document databases. It would even be possible to query the documents solely on the basis of meta-data (such as "find all HTML pages with more than 3 images."). An SQL query (assuming we have a database called WWW containing pairs of URLs and their HTML content) to process this would simply be:

```
SELECT URLPair(URL, count(HTML..IMG)) from WWW
GROUP BY URL
HAVING count(HTML..IMG) > 3
```

In this paper, we present formalisms that would enable us to use queries like the above on document databases. To facilitate this task, we first describe our model of structured documents, and then introduce a formal calculus language which provides the basis for the above language. We then describe an algebra to demonstrate the operations, and finally the SQL-like language. We describe DocBase, a research system that implements the ideas presented here. Here wee will only present an overview of the methodologies used in the design DocBase. Further details on these methods can be obtained from [Sen97][1].

## 2 Searching and querying

The most common method for searching information in a document repository is by using *boolean searches* [Sal91]. In this type of search, a number of keywords combined with boolean operators (such as "and", "or", "not") are specified, and the result consists of the documents that satisfy the given boolean expression. The problem with this type of search is that all matching documents in the resulting document set are given the same importance. To avoid this, one can use the *weighted keyword search*, in which the search items are assigned weights based on their importance, and the retrieved documents can be ordered by relevancy based on the number of matches and the weights of the matched terms.

The data extraction mechanism in databases differs considerably from that in information retrieval systems. Searches in databases always involve meta-data. It is trivial to pose database queries

[1] A copy of this thesis can be obtained online from http://www.cs.indiana.edu/~asengupt/thesis-final.ps.gz

using only meta-data. However, formulating meta-data independent queries in database systems is quite difficult, and is a subject of recent research[JMG95].

### 2.1 Unifying the worlds - document database systems

Research on the problem of using structured documents has produced a few methods for querying structured documents. Most of these methods (such as [CACS94, Zha95, Hol95, D'A95]) use various mapping techniques to map document structures into database schema, and import the documents into database objects of a host database. The capabilities of the host database can then be used to perform queries. Although this method works fairly well, it does have many drawbacks. The conversion process is usually expensive, and the incompatibilities between document structures and database schema often result in loss of information. Moreover, this method requires that the documents be replicated in the host database system. Since documents can often be large and distributed in many different formats and media, replicating these documents from their different formats into the database is sometimes too wasteful, and often not practical.

## 3 A formal structured document model

Documents do not fit well into traditional database models because of their complex hierarchical structure. An attempt to model document structures in relational databases causes too much fragmentation in the documents, since every parent-child relationship in the documents results in a separate table. Complex-object and object-oriented models can be more appropriately used to represent structured documents. However, structured documents form a sufficiently self-contained domain to consider a model for themselves. In this paper, we consider an independent model for documents keeping in mind the underlying SGML model.

### 3.1 Related approaches

Document modeling is a fairly well-researched problem in databases. Documents have been typically modeled in three prominent ways:

1. *Unstructured sequence of text:* In this model, documents are treated as a long sequences of text (the granularity of characters or words). Searches are performed primarily by matching keywords against the text, using indices as a means for achieving higher search speed. Research on unstructured text searching falls under the discipline of "information retrieval (IR)". Prominent work in this area include the foundations from Salton[SYY75, SB88, Sal91] and also many implementation methods such as inverted indices [MW93, TGMS94]. Our approach differs from these approaches since we actually use the structural information in the documents as meta-data for query processing.

2. *Semi-structured data with nodes and links*: In this model, documents are represented as graphs with nodes and links, possibly with inconsistent or unknown structure. Labels at the nodes or edges in the graph represent the data and meta-data in the documents, and query processing involves traversal and transformation of the graphs. Prominent among these approaches is the work by Buneman et al.[BDHS96]. Many other related approaches can be found in [Suc97]. While this approach comes fairly close to the structured document domain, the emphasis is still on mainly unstructured documents. Our approach intends to formalize a method for fully structured documents using the schema information embedded in the structure.

3. *Fully structured data with known structure schema*: In this model, documents are treated as having a well-defined structure. This approach typically models document structure using a grammar and documents as words in the language described by the grammar. Querying methods in this approach include operations based on the grammar which governs the database schema [DGS86, GT87, Col89]; using existing database systems to implement an equivalent database schema to store the documents; and using special structures such as Patricia trees [GBY91]. In our approach we do not require a conversion of the documents to a host database. However, we do use indices to speed up queries.

### 3.2 The document model used in DocBase

To define the notion of structured document databases, we first define a few sets: **gi** is a countably infinite set of generic identifiers (GIs); **doc** $\subseteq$ **gi** is the set of document types (we describe later the reason for making this distinction), and **att** is a countably infinite set of SGML attributes. We also define **dom**, which is a countably infinite set of constant character strings, and **var**, a countably infinite set of variables.

**Types** We only consider two types:

1. *Basic type* $\beta$. The base type comprises of character strings, with **dom** being the range of values.

2. *Complex type* $\tau$. The complex types form the set **gi**. Within this set, the subset **doc** defines document types which are special complex types in the database. Document types are considered to be special complex types because they define the type of a complete document for a given DTD. However, any GI in a DTD defines a complex type, and new DTDs may be constructed starting at that particular GI, thus making it a document type.

**Documents and databases** Documents form the core component in a document database system. We define documents and databases consisting of documents as follows:

- *Document.* Intuitively, a document is an SGML instance of a DTD. A DTD can be modeled as a grammar represented by a quintuple $d = (\tau, \mathcal{G}, \mathcal{A}, \mathcal{C}, \mathcal{P})$ where $\tau \in$ **doc** is a document type, $\mathcal{G} \subset$ **gi** is a set of generic identifiers, $\mathcal{A} \subset$ **att** is a set of SGML attributes and $\mathcal{C} \subset$ **dom** is a set of constants. $\mathcal{P}$ is a set of production rules describing the structure of conforming document instances. Analogous to the relational database model, the DTD serves as the schema for the managed data, and documents conforming to the DTD serve as instances of the schema.

- *Database. A database, in this setting, is a finite set of SGML documents conforming to one of the document type definitions in* **doc**.

## 4 A formal query language for structured documents

We now briefly describe a calculus for documents. This calculus is an extension of relational calculus supporting path expressions and operations on document structures. To reduce the complexity of the query language, we use a simplified path expression construct. We then define the language by defining the terms, operators, predicates, formulas, and finally, queries in the language.

## 4.1 Path expressions

The notion of path expressions (PEs) came from two different areas: (i) graph query languages and (ii) object-oriented query languages. For graph query languages (*e.g.,* [MW95]), a path expression defines a path from one node in the graph to another in terms of intermediate node and edge labels. For object-oriented query languages (*e.g.,* [KKS92, dBV93]), a path expression defines a path from one object to another using membership and inheritance relationships. We propose a simplified path expression construct specifically for structured documents.

**PEs in document structure** Path expressions for document structures have also been considered by Christophides et al. [CACS94, CCM96]. These approaches allow variables on paths, which instantiate over paths in the current domain. Christophides et al. also propose a syntactic sugared path expression of the form $my\_article..title(t)$ when the actual values of path variables are not important. Formally, the above expression evaluates to:

$$\exists p \ \text{PATH}(p) \wedge my\_article.p.title(t)$$

Here, the expression indicates that there is a path between $my\_article$ and $title(t)$, but the actual path itself is not of significance.

**Simple Path Expressions (SPE)** We are interested primarily in posing queries in document databases. Although completely generalized path expressions with path variables and selectors increase the expressive power of a language, they are rarely used in a document context, especially for SGML documents. Further, document structures in SGML are strictly hierarchical (*i.e.,* they form a tree rather than a graph) and although SGML allows recursion of elements, well-designed document structures are usually non-recursive. In addition, our path expressions are only used for structure traversal rather than link chasing. With these observations, we now define the notion of simple path expressions (relative to the DTD $D$), as follows:

- *Null path.* $\epsilon$, the null path, is an SPE.
- *Basic path.* Every $x \in$ **gi** is an SPE, and is termed "basic path."
- *Listed path*: A listed path $\mathcal{P}$ is a fully expanded SPE of the form $\mathcal{A}_1.\mathcal{A}_2. \cdots .\mathcal{A}_n$, where $n \geq 1$, and for every $i$, $\mathcal{A}_i \in$ **gi**, and $\mathcal{A}_{i+1}$ is in the *content group*[2] of $\mathcal{A}_i$.
- *Abbreviated path.* An abbreviated path $\mathcal{P}$ of the form $\mathcal{P}_1..\mathcal{P}_2.. \cdots ..\mathcal{P}_k$ is an SPE, where $k \geq 1$, and for every $i$, $\mathcal{P}_i$ is a basic or listed path. This notion is similar to the partial path specification in [dBV93] and the ".." operator in [CACS94], described above.

**Semantics of SPEs** As mentioned above, path expressions are always defined in the context of a DTD. A path expression $\mathcal{P}$ applied to a set of documents is a function from a set of documents to another set of documents rooted at $last(\mathcal{P})$, such that these new sets of documents are rooted at elements that are reachable from the roots of the original documents by following the path $\mathcal{P}$. Here, $last(\mathcal{P})$ refers to the last GI in $\mathcal{P}$.

[2] In SGML, the declaration of a generic identifier is written as: `<!ELEMENT thisgi - - (G1, G2, G3...)>`. The GIs within parenthesis are the children of thisgi, and are said to be in thisgi's content group.

## 4.2 Closure

In addition to using SGML for the storage and modeling of the data, one primary issue in this research is to achieve closure in the domain of SGML. Simply put, this means that SGML is to be used as the input as well as the output format for queries. Closure is prominent in the relational database model as tables are both inputs to queries and outputs from queries. Closure is important because it allows nesting of queries and the concept of views, and also because it simplifies the implementation of the input-output handling of such languages. In our work, all query languages are closed under our document model, in the sense that they all take valid SGML documents as input and produce valid SGML documents as output. In addition, the DSQL DTD language that we propose in Section 5.2 takes this concept of closure one step further by making the query language itself a valid SGML document. We will discuss the effects of this extended closure in that section.

## 4.3 The Document Calculus (DC)

We now describe Document Calculus (DC) as a calculus for documents. DC is an extension of the relational calculus, and uses SPEs as terms. We will use the symbol $\tau$ for types, and the symbol $\circ$ to represent one of the path expression operators **.** and **..** in the following discussion.

**Terms** Terms in DC include constants ($c \in$ **dom**); variables ($x_\tau \in$ **var**, $\tau \in$ **gi**); and path terms ($x \circ \mathcal{P}$) where $\circ \in \{.\ ,\ ..\}$.

**Operators** Basic comparison operators and logical operations are supported in this language. Comparison operators include atom to set comparison operators ($\ni$ and $\not\ni$) and set to set comparison ($=, \neq, \subseteq, \subsetneq, \cap, \not\cap$). The last two operators are intersection operators, *i.e.*, $R \cap S$ is true if R and S intersect. In addition, DC supports the standard logical operators $\wedge$ (AND), $\vee$ (OR), $\neg$ (NOT).

**Predicates** The predicates supported are document predicates such as $D \in$ **doc**, path and path term predicates (of the form $D \circ \mathcal{P}$ or $x \circ \mathcal{P}$).

**Formulas** DC formulas are functions from a set of variables to the boolean values `true` and `false`. The formulas in DC include the following:

1. *Atomic formulas.* $\mathcal{R}(x)$ is an atomic DC formula, where $\mathcal{R}$ is a predicate.
2. $x \circ \mathcal{P} \theta c$ is a DC formula, where $\theta \in \{\ni, \not\ni\}$, $x \circ \mathcal{P}$ is a path term and $c \in$ **dom** is a constant.
3. $t_1 \theta t_2$ is a DC formula, where $\theta \in \{=, \neq, \subseteq, \subsetneq, \cap, \not\cap\}$ and $t_1 = x_1 \circ \mathcal{P}_1$ and $t_2 = x_2 \circ \mathcal{P}_2$ are two path terms.
4. If $\varphi$ and $\psi$ are formulas, so are $\varphi \vee \psi$, $\varphi \wedge \psi$, $\neg\varphi$.
5. If $\varphi(x, x_1, x_2, \ldots, x_n)$ is a DC formula with $n+1$ free variables $x, x_1, x_2, \ldots x_n (n \geq 1)$ then the following are DC formulas:
   - $\exists x\ \varphi(x, x_1, x_2, \ldots, x_n)$ (existential quantifier).
   - $\forall x\ \varphi(x, x_1, x_2, \ldots, x_n)$ (universal quantifier).
6. If $\varphi$ is a formula, so is $(\varphi)$.

Ideally formulas are useful if only a finite number of such sets of values satisfies the formula. However, in the above setting of formulas, it is not possible to guarantee that only a finite combination of the free variables satisfy the formula. For example, the query "all documents not in the database" can be represented by the formula $\neg D(x)$ and can be satisfied by an infinite number of values of $x$. Such formulas are unsafe, since queries that include such formulas can never be computed in a finite time. To avoid this problem, we also define the notion of *safe formulas*.

**Safe DC formulas** Safe DC formulas (or, in short, SDC formulas) are the formulas which can be satisfied by only a finite set of values for the free variables. This is achieved by ensuring that the values of all free variables are always restricted to finite sets and by ensuring that potentially unsafe operations (such as negation, as in the example above) always occur along with another formula that restricts the selection of values of the free variables. To achieve this, every variable is associated with a set that binds the possible values for that variable (*i.e.*, instead of $x$, we would write $x^{\psi}$ where $\psi$ is a finite precomputed set of documents). For the complete definition of Safe DC formulas, please see [Sen97].

**Tuple Construction** The path expressions provide a means for extracting components of a composite object. DC supports dynamic creation of composite types by creating new generic identifiers with already existing generic identifiers as children. This is achieved by providing a tuple construction expression of the form:

$$z = R\langle x_1, x_2, \ldots, x_n \rangle \quad n \geq 1$$

The tuple construction operation can be treated as a formula that always returns a truth value, and thus can be combined with other formulas using a conjunction.

**Queries** A query is an expression denoting a set of documents described by a safe DC formula $\varphi$. Queries give the closure property to the language by ensuring that the output of the query will be valid documents. All queries in DC are of the form

$$\{x \mid \varphi(x)\}$$

Here we only consider formulas with a single free variable, since such a formula can always be constructed from any formula with multiple free variables using the tuple construction mechanism described above, as follows:

$$\varphi_1(z) \triangleq (z = R\langle x_1, x_2, \ldots, x_k \rangle) \wedge \varphi(x_1, x_2, \ldots, x_k)$$

**An illustration** To illustrate the language, consider the schema in Figure 1.

```
<!DOCTYPE POEM [
<!ELEMENT poem       - - (head, body)>
<!ELEMENT head       - - (period, poet, title)>
<!ELEMENT body       - - (stanza)+>
<!ELEMENT stanza     - - (line)+>
<!ELEMENT (period | poet | title | line) - O (#PCDATA)>
]>
```

Figure 1: A simple poem database schema

Ex. 1. The query 'Find all poems that contain the word "love" in the poem title' would be written in DC as $\{x|x^{\psi}..title \ni \text{"love"}\}$. Here, $\psi$ is given by the query $\{z|poem(z)\}$.

Ex. 2. Consider the query 'Find the pairs of names of poets who have at least one common poem title'. Suppose $\psi = \{z|poem(z)\}$. The query is then expressed as:

$$\left\{v \mid (v = R\langle w, z\rangle) \wedge (\exists x, y\ (x^{\psi}..title \cap y^{\psi}..title \wedge x^{\psi}.\epsilon \not\cap y^{\psi}.\epsilon \wedge x^{\psi}..poet(w) \wedge y^{\psi}..poet(z))\right\}$$

# 5 Practical query languages

DC is useful for describing the properties of our document model and the types of queries we are interested in. DC is a language like relational calculus with the additional capabilities of tuple constructions, and navigational constructs with path expressions. In [Sen97], we show many useful properties of this language, including safety and PTIME properties. In order to prove these properties, as well as to provide a means for the implementation of this language, we propose an algebraic language that we call Document Algebra (DA) and show that DA and SDC are equivalent. In addition, we show two other equivalent languages: (i) DSQL (Document SQL), an SQL-like language with the same capabilities as DC, and (ii) DSQL DTD, an SGML document type for a language that enables queries to be written in SGML itself. Here we briefly describe these three languages.

## 5.1 The Document Algebra (DA)

DA is predictably an extension of the relational algebra. As any algebraic language, DA is defined in terms of special set operators that map one or more sets of documents to a new set of documents. One important property of DA is that it always produces documents that adhere to our original notion of documents, *i.e.*, always conforming to specific document types. To achieve this, operations in DA may add one or more productions to the existing DTD. Further, all operations in DA are closed under the SGML domain - they all generate document instances as their output.

Every DA expression $E^{\tau}$ represents a set of documents of a particular type $\tau$. We define DA expressions inductively by first defining the basic document expression and then defining the operations *path selection ($\circ$), cross product ($\times$), selection ($\sigma$), union ($\cup$), intersection ($\cap$) and set difference ($-$).* Other useful operations such as *join* ($\bowtie$), projection ($\pi$), generalized product ($\prod$) and root addition ($\rho$), can be constructed from these primitive operations. All the primitive operators generate documents of specific types, as shown in Table 1.

| Expression | Type | New productions |
|---|---|---|
| $D$ | $D$ | |
| $E \circ \mathcal{P}$ | $last(\mathcal{P})$ | |
| $E_1^{\tau_1} \cup_R E_2^{\tau_2}$ | $R$ | $R \rightarrow \tau_1 \mid \tau_2$ |
| $E_1^{\tau} - E_2^{\tau}$ | $\tau$ | |
| $E_1^{\tau_1} \times_R E_2^{\tau_2}$ | $R$ | $R \rightarrow \tau_1, \tau_2$ |
| $\sigma_{\gamma} E^{\tau}$ | $\tau$ | |

Table 1: Types of Document Algebra operations and new created types.

Intuitive explanation of each of the above operators is as follows:

**Document** The expression $D$ represents the set of all documents in the database.

**Path selection** ($\circ$) Given a DA expression $E^\tau$ and a SPE $\mathcal{P}$, $E^\tau \circ \mathcal{P}$ is a DA expression that returns the set of documents rooted at $last(\mathcal{P})$ obtained by traversing the path $\mathcal{P}$ from each of the documents in $E^\tau$.

**Union** ($\cup$) Union is the usual set union operation, without the restriction that both operands of the union be of the same type. Given two DA expressions $E_1^{\tau_1}$ and $E_2^{\tau_2}$, the result of the union $E_1^{\tau_1} \cup_R E_2^{\tau_2}$ is a set of documents of a new type $R$ which is created by adding a production for $R$ with $\tau_1$ and $\tau_2$ in an option group.

**Intersection** ($\cap$) The intersection operation is the usual set intersection operation $E_1^\tau \cap E_2^\tau$, containing the set of documents that in both $E_1^\tau$ and in $E_2^\tau$.

**Set difference** ($-$) The set difference is the usual set difference operation $E_1^\tau - E_2^\tau$, containing the set of documents in $E_1^\tau$ not in $E_2^\tau$.

**Cross product** ($\times$) Given two DA expressions $E_1^{\tau_1}$ and $E_2^{\tau_2}$, the expression $E_1^{\tau_1} \times_R E_2^{\tau_2}$ is a DA expression, and it represents a set of documents with a new type $R$, the members of which contain two subcomponents: one from the set $E_1^{\tau_1}$ and the other from $E_2^{\tau_2}$. Every member in the resulting set is hence of type $R$, and each member of the set $[\![E_1^{\tau_1}]\!]^{\mathbb{M}}$ is paired with each member of the set $[\![E_2^{\tau_2}]\!]^{\mathbb{M}}$ under an instance of $R$.

**Selection** ($\sigma$) The selection operation $\sigma_\gamma E^\tau$ extracts a subset of documents from the input set $E^\tau$ that satisfy a selection condition $\gamma$.

We can again consider the example queries we used before:

Ex. 1. The first example is intuitively: $\sigma_{poem..title \ni \text{"love"}} Poem$.

Ex. 2. The second example clearly requires a cross product between poem and itself. To simplify the expression, we use the derived projection, join and root addition operators.

$$\pi_{P_1..poet, P_2..poet} \left( \sigma_{P_1..poet \not\sim P_2..poet} \left( \rho_{P_1} Poem \bowtie^R_{P_1..title \cap P_2..title} \rho_{P_2} Poem \right) \right)$$

Given the above specification of DC and DA, we have proved the following significant results (the proofs are beyond the scope of this paper, please see [Sen97] for the actual proofs):

**Theorem 1** The languages DA and SDC are equivalent.

**Theorem 2** The languages DA and SDC are safe.

**Theorem 3** The languages DA and SDC are in PTIME.

## 5.2 DSQL - an SQL-like language

Although DA can be used to adequately provide for an implementation of, as well as show safety and complexity of the language, it is still not useful as a language for interactively processing queries. We now describe *DSQL* (Document SQL)[3], an extended version of SQL which is a user-friendly pseudo-natural language form of DC. An informal introduction and examples of this SQL can be found in [Sen96], and the full language with BNF is presented in [Sen97]. The primary motivation

[3]Note that DSQL (or Document SQL) is different from SDQL (Standard Document Query Language), which is a part of the ISO 10179 DSSSL (Document Style Semantics and Specification Language) standard [ISO94].

behind having such a language is to provide users of database systems with a simple means for expressing queries using a natural language form. Also, SQL's wide acceptance as a standard query language for relational databases makes it natural choice as a document database query language. DSQL is designed as an extension to the standard SQL-86 [SQL86]. Conceptually DSQL supports SGML documents as objects for constructing queries. From the language point of view, however, there are only two major differences from the standard SQL, which are the following:

1. *Path Expressions.* Path expressions are handled in the same way they are handled in the formal languages. To use path expressions, two main changes are made to SQL. The standard "**.**" operator used commonly in SQL to denote relation attributes can now be cascaded to express listed paths. In addition, a "**..**" operator is introduced, which is used to construct an abbreviated path from a listed path.

2. *Complex selections.* Standard SQL deals with flat tables as primary objects, and hence specifies the output as a set of columns that constitutes the output table. In DSQL, the primary objects on which queries are built are documents. To ensure closure, output of queries is also specified using document formulation constructs. To accommodate this feature, the select clause allows the creation of composite document types from constituent components. This is similar to the tuple construction operation in DC, using which, new types are created.

Consider the example queries as before:

Ex. 1. The first query is simple:

```
SELECT P
FROM POEM P
WHERE P..title = "love"
```

Ex. 2. For the second query, we need two instances of POEM:

```
SELECT R(P1..poet,P2..poet)
FROM poem P1, poem P2
WHERE P1..title = P2..title
AND P1..poet <> P2..poet
```

Details on DSQL and further examples can also be obtained from [Sen97].

## 5.3 Uniting SGML and SQL: the DSQL DTD

We can describe SGML as a meta-language which can define languages which, in turn, define valid document instances. Thus, SGML can be conveniently used to define a query language. The DSQL language has a syntax that can be conveniently expressed using a BNF, and hence, using an SGML DTD. With this observation, we also defined a DTD for DSQL, using which DSQL queries can be written in SGML itself. There are a few distinct advantages of using SGML as a query language:

- First and foremost, this query language retains the properties of both SQL and SGML. Being an application of SGML, this language is inherently portable and is independent of the underlying system and platform. On the other hand, since it is equivalent to DSQL, the DSQL DTD defines a first order, low complexity query language.

- Since queries are in SGML, which is the same data format as the database itself, the queries can be stored and managed in the same way as the data itself, thus exhibiting a strong closure property. This immediately implies some interesting possibilities:
  - Queries can be stored as data and, subsequently, can be queried themselves to extract information that will be very suitable in applications such as data mining and performance tuning.
  - The capability of storing queries as data allows subsequent treatment of data as queries. This ability is commonly known as "reflection" in programming languages, and gives a language a higher expressive power and the capability of performing meta-data queries.
- Queries formulated and stored in SGML can easily be converted into any other query language (including visual query languages) without much effort.
- Users posing queries in SGML can do so within their familiar environment of SGML editors. This capability also ensures that they do not have to learn the syntactic details of a new language, and a validating editor will ensure that all the queries are valid DSQL queries.
- SGML queries can be seamlessly integrated within other SGML documents (possibly using the SUBDOC feature of SGML) for dynamic document content. Queries embedded in a document can be replaced by the results obtained from the queries before presenting the final document. This is a natural way of dynamic document content generation for the WWW.

In this language, the first example query[4] as before, will be written as:

```
<!DOCTYPE SQL PUBLIC "-//DocBase//DTD DSQL//EN''>
<select>
  <output><scalar><col><pathlist><gi>poem</pathlist></col></scalar>
  </output>
<from><db><pathlist><gi>poem</pathlist></db></from>
<where><cond><predicat><compare EQUAL>
   <scalar><col><pathlist><gi>poem</pathlist>
      <pathlist><gi>title</pathlist></col></scalar>
   <scalar><atom>"love"</scalar>
</predicat></cond>
</where>
</select>
```

# 6 User interface issues

The success of any system depends not only on the features of the system but also on its "usability." Even if a system is feature-rich, such features are useless if they cannot be easily accessed by the users. "User interface" is a generic term given to the way a system interacts with its users. To design usable systems, the design process needs to incorporate usability considerations into the early stages. In the design of DocBase, we also gave careful consideration to user interface issues, and the result was a visual interface to our query languages, called QBT (Query By Templates). Here we briefly describe QBT. Details on QBT can be obtained from [Sen97, SD97].

[4]Because of space limitations, we are skipping the second example. Readers please note that this example is essentially equivalent to the DSQL version shown earlier.

### 6.1 QBT: A visual query language

QBT is a generalization of an early visual query language for relational database, called QBE (Query By Example)[Zlo77]. QBE is suitable for relational databases since it uses tabular skeletons (analogous to tables in the relational model) as a means for constructing queries. Thus, the template for presenting queries in QBE is similar to the conceptual structure of the instances in the database. We use this idea to generalize QBE for databases where each data instance, albeit complex, has a simple visual model. We base this assumption on the fact that human beings form a mental model for the tasks that they intend to perform [Boo89]. For example, users performing a search in a dictionary may not know the internal structure and representation of each definition, but they usually have an idea about the visual structure of a dictionary entry, assuming they have used dictionaries in print. In our method, the basis of the interface is a visual template representing an instance of the database. Simple examples of templates include (i) a small poem for poetry databases, (ii) a table for relational databases, (iii) a representative word definition for a dictionary database, and (iv) a sample entry in a bibliography database.

QBT is primarily designed to be a simple point-and-click interface for posing queries in document databases without knowing or understanding the internal structure of the database and without learning complex query language syntax. In spite of its apparent simplicity, QBT is a powerful language and can express the same class of queries as the core DSQL language (DSQL without any nested queries).

### 6.2 Design of QBT

A QBT interface, in its simplest manifestation, displays a template for a representative entry of the database. The user sees a sample of the type of data she would expect to find in the database (*e.g.*, a poem in a poetry database). She specifies a query by entering examples of what she is searching for in the appropriate areas of the template, and the system retrieves all the database entries that match the example she provided. To illustrate the interface, we will use a simple template for a poetry database, as in Figure 2. In this figure, we indicate a prominent logical region of the poem by circling it and labeling it with the corresponding region name. Physically, the QBT interface consists of a small template image divided into areas corresponding to the different logical regions in the database, as in Figure 2.

Based on the nature of the placement of the logical regions on the physical areas of the template, the templates can be classified into two categories: (i) flat templates (physical regions do not overlap, as in Figure 2) and (ii) nested templates (one or more physical regions are totally inside one physical region). Nested templates can be visualized by either showing the nested regions on the same template, or using a "zooming in" method to magnify a region when selected, if the region has embedded regions. We show in [Sen97] that we can perform many different types of queries including joins, and also show that the class of queries representable by QBT is the same as that of the core DSQL.

## 7 DocBase - a modular document database

With the components of a conceptual document database system described above, we implemented a prototype system called DocBase[5]. We developed DocBase with the idea that most users dealing with electronic documents, especially structured electronic documents, already have a means for

[5]More details and demonstrations about DocBase are available in http://www.cs.indiana.edu/~asengupt/docbase/

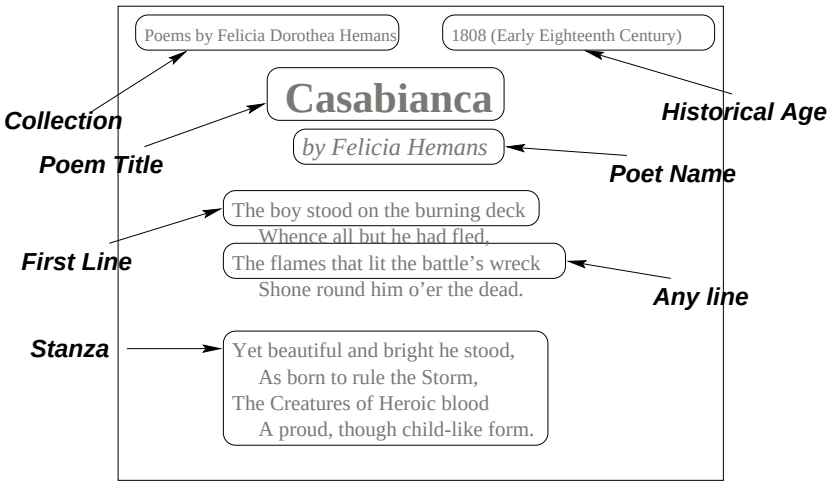


Figure 2: A simple template for poems, with its logical regions

editing and updating these documents. The requirement of "querying in place" is actually driven by this knowledge. In addition, users may already have indexing and searching capabilities on their documents (possibly in information retrieval style). We designed DocBase so that users can essentially "plug and play" with different types of indexing systems for search purposes and different types of storage managers for storing and controlling documents. We have actually tested the system with two storage managers (Exodus [CDF+86] and Sybase [Syb94]), and presently only one indexing system (Open Text Pat [Ope94]) - however, work is under way to include another structured search system called "Sgrep"[JK96] as the index manager.

The architecture of DocBase closely follows the tri-level design of database systems. In this architecture, there are three distinct layers: (i) a top layer involving interaction with the user, (ii) a middle layer involving query parsing, translation and optimization, and (iii) a bottom layer involving actual processing of the query (see Figure 3). In the current implementation, the top layer of the system comprises of a QBT or a command line SQL interface. The query processing layer consists of (i) a parser (written using lex and yacc [LMB92]) which identifies each component of the query, (ii) an optimizer or reorganizer which generates an execution plan, and (iii) a query engine which actually executes each of the query components, using any available indices.

The query engine often needs to create indices on the fly based on the type of the query (join queries in particular). These indices and temporary results are stored in the storage manager. The query is processed in an accumulator-based evaluation (the accumulators in this case are simply the temporary results at each stage). The details of the processing algorithms are discussed in [Sen97]. The accumulators can be viewed as the intermediate results from the evaluation of the component queries.

The modular nature of the system is achieved by using an standard interface for storage management and index management modules. To add the support of a different storage manager or indexing system, a subclass of the virtual storage manager or index manager needs to be created, with the individual methods fully implemented according to the specifications in the virtual classes. Finally, the system can be adapted to use the newly added class with a configuration option.

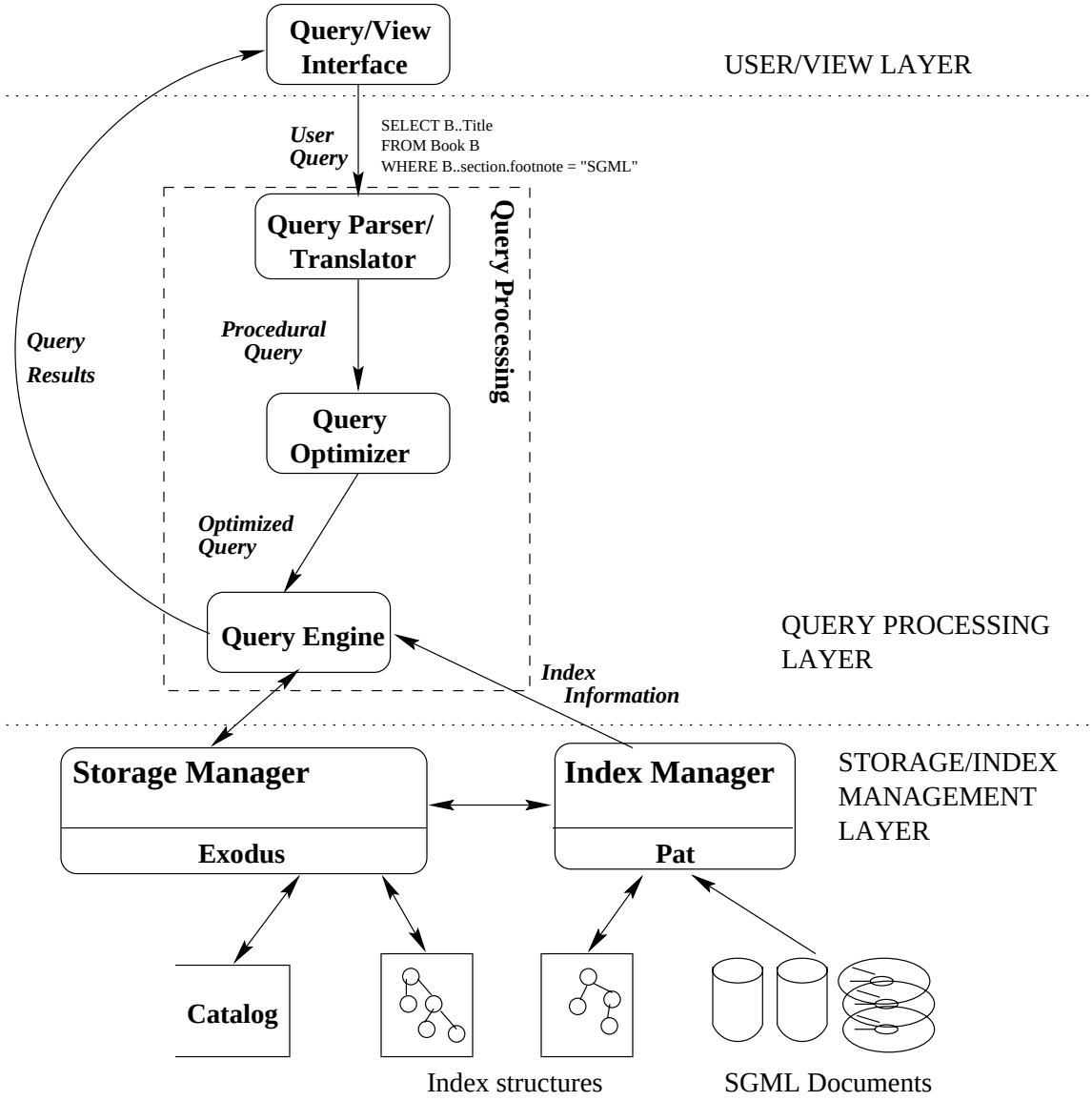


Figure 3: The architecture of DocBase

### 7.1 Query Optimization

Design of extensive query optimization techniques in this evaluation method is still an object of future research. The main problem here is to determine the semantics of the optimization operations in this setting. Since most of our operators were obtained by simple extensions to relational operators, most commutative rules still hold in our setting, which implies that most relational query optimization methods can be easily applied to this evaluation scheme. The current implementation of DocBase already optimizes selections by performing them as early as possible, and using indices whenever possible. In addition, DocBase has some simple path expression optimization schemes that reduces the amount of structural traversal for the evaluation of the path queries.

## 8 Conclusion

With HTML already in full flow and XML coming around the corner, the next century clearly shows a trend towards the use of structured documents for document interchange. SGML is already being used heavily for extensive documentation and many research and commercial publications. The need clearly exists for a system-independent way of searching these documents, instead of developing system-specific search methods for every application. Our proposal for a SGML document model with its corresponding query languages, and a framework for its implementation is the most important contribution of this research. The following are other significant contributions of this work:

1. Proposal for a formal model for structured documents,
2. Proposal for a system of equivalent low-complexity query languages,
3. Use of SGML itself as a query language for SGML,
4. Development of an architecture for query processing on SGML documents *in place*, and
5. Development of a generalized visual interface for document queries.

Traditionally databases have always been viewed as highly structured discrete sets of data. Although databases may contain extensive textual information in text or "memo" fields, such information has not been efficiently searchable and indexable. On the other hand, information retrieval systems have shown us how to efficiently search text, but has made little use of embedded structure in document, even if available. In this work, we have united these two fields towards document database systems of the forthcoming century. SGML's generalized nature and ability to represent many types of data allows its use in domains other than document databases, and the prospect of using SGML database systems in many future database applications is highly likely in the forthcoming century.

## References


[BDHS96] Peter Buneman, Susan Davidson, Gerd Hillebrand, and Dan Suciu. A query language and optimization techniques for unstructured data. In H.V. Jagadish and Interpal Singh Mumick, editors, *Proceedings, ACM SIGMOD International Conference on Management of Data*, volume 25, pages 505–516, June 1996.

[Boo89] Paul Booth. *An Introduction to Human-computer Interaction*. Laurence ErlBaum Associates Publishers, 1989.

[CACS94] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. *SIGMOD RECORD*, 23(2):313–324, June 1994.

[CCM96] Vassilis Christophides, Sophie Cluet, and Guido Moerkotte. Evaluating queries with generalized path expressions. In H.V. Jagadish and Inderpal Singh Mumick, editors, *Proceedings, ACM SIGMOD 1996*, volume 25, pages 418–422. Association of Computing Machinery, June 1996.

[CDF+86] Michael J. Carey, David J. DeWitt, Daniel Frank, Goetz Graefe, M. Muralikrishna, Joel E. Richardson, and Eugene J. Shikita. The architecture of the EXODUS extensible DBMS. In Klaus R. Dittrich and Umeshwar Dayal, editors, *Proceedings, 1996 International Workshop on Object-Oriented Database Ssytems*, pages 52–65, Pacific Grove, California, USA, September 23-26 1986. IEEE-CS.

[Col89] Latha S. Colby. A recursive algebra for nested relations. Technical Report 259, Indiana University, January 1989.

[D'A95] Al D'Andrea. Improved database technology for document management. In Yuri Rubinsky, editor, *Proceedings, SGML '95*, pages 113–122. Graphic Communications Association, December 1995.

[dBV93] Jan Van den Bussche and Gottfried Vossen. An extension of path expressions to simplify navigation in object-oriented queries. In Stefano Ceri, Katsumi Tanaka, and Shalom Tsur, editors, *Proceedings of the third international conference on Deductive and Object-Oriented Databases (DOOD)*, number 760 in Lecture Notes in Computer Science, pages 267–282, Phoenix, Arizona, December 1993. Springer-Verlag.

[DGS86] B.C. Desai, P. Goyal, and F. Sadri. A data model for use with formatted and textual data. *JASIS*, 1986.

[GBY91] Gaston H. Gonnet and R. Baeza-Yates. Lexicographical indices for text: Inverted files vs pat trees. Technical Report TR-OED-91-01, University of Waterloo, 1991.

[GT87] Gaston H. Gonnet and Frank W. Tompa. Mind your grammar: a new approach to modeling text. In Peter M. Stocker, William Kent, and Peter Hammersley, editors, *Proceedings: 13th International Conference on Very Large Data Bases*, pages 339–346, Brighton, England, September 1-4 1987. Morgan Kaufmann.

[Hol95] Sebastian Holst. Database evolution: the view from over here (a document-centric perspective). In Yuri Rubinsky, editor, *Proceedings, SGML '95*, pages 217–223. Graphic Communications Association, December 4-7 1995.

[ISO86] International Organization for Standardization, Geneva, Switzerland. *ISO 8879: Information Processing – Text and Office Systems – Standard Generalized Markup Language (SGML)*, 1986.

[ISO94] International Organization for Standardization and International Electrotechnical Commission, Geneva, Switzerland. *ISO/IEC DIS 10179: Document Style Semantics and Specification Language: DSSSL*, 1994.

[JK96] Jani Jaakkola and Pekka Kilpeläinen. The sgrep online manual. Available online at http://www.cs.helsinki.fi/ jaakkol/sgrepman.html, 1996.

[JMG95] Manoj Jain, Anurag Mendhekar, and Dirk Van Gucht. A uniform data model for relational data and meta-data query processing. In *Proceedings of the Seventh International Conference on Management of Data (COMAD)*, pages 146–165. Tata McGraw-Hill Press, December 1995.

[KKS92] Michael Kifer, Won Kim, and Yehoshua Sagiv. Querying object-oriented databases. In Michael Stonebraker, editor, *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, pages 393–402, June 1992.

[LMB92] John R. Levine, Tony Mason, and Doug Brown. *Lex & yacc*. O'Reilly & Associates, 2nd ed. edition, 1992.

[MW93] Udi Manber and San Wu. Glimpse: A tool to search through entire file systems. Technical Report TR 93-34, University of Arizona, October 1993.

[MW95] A.O. Mendelzon and P.T. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computing*, 24(6):1235–1258, December 1995.

[Ope94] Open Text Corporation, Waterloo, Ontario, Canada. *Open Text 5.0*, 1994.

[Sal91] Gerard Salton. Developments in automatic text retrieval. *Science*, 253:974–980, 1991.

[SB88] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24:513–523, 1988.

[SD97] Arijit Sengupta and Andrew Dillon. Query by templates: A generalized approach for visual query formulation for text dominated databases. In Alfred Aho, editor, *Proceedings: Symposium on Advanced Digital Libraries*, Library of Congress, Washington, DC, May 7-9 1997. IEEE/CESDIS.

[Sen96] Arijit Sengupta. Demand more from your SGML database! bringing SQL under the SGML limelight. *<TAG> The SGML Newsletter*, 9(4):1–7, April 1996.

[Sen97] Arijit Sengupta. *DocBase - A Database Environment for Structured Documents.* PhD thesis, Indiana University, December 1997.

[SQL86] ANSI X3.135-1986, Database Language SQL, 1986.

[Suc97] D. Suciu, editor. *Proceedings on the Workshop on Semistructured Data*, Tucson, Arizona, USA, May 1997.

[Syb94] Sybase, Inc., Emeryville, CA. *SYBASE SQL Server$^{TM}$ Reference Manual: Volume 1. Commands, Functions and Topics*, 1994.

[SYY75] G. Salton, C.S. Yang, and C.T. Yu. A theory of term importance in automatic text analysis. *Journal of the American Society of Information Science*, 26(1):33–44, 1975.

[TGMS94] Anthony Tomasic, Hector Garcia-Molina, and Kurt Shoens. Incremental updates of inverted lists for text document retrieval. *SIGMOD RECORD*, 23(2):289–300, June 1994.

[W3C97] W3C. *Extensible Markup Language (XML) W3C Working Draft 07-Aug-97*, August 7 1997. Available on-line from http://www.w3.org/TR/WD-xml-lang.

[Zha95] Jian Zhang. Oodb and sgml techniques in text database: An electronic dictionary system. *SIGMOD RECORD*, 24(1):3–8, March 1995.

[Zlo77] M. M. Zloof. Query by example: A database language. *IBM Systems Journal*, 16(4), 1977.