Distributed Query Processing for Structured and Bibliographic Databases

Ee-Peng Lim
School of Applied Science
Nanyang Technological University
Nanyang Ave., Singapore 639798
aseplim@sentosa.sas.ntu.ac.sg

Ying Lu
Institute of Systems Science
National University of Singapore
Heng Mui Keng Terrace, Kent Ridge, Singapore 119597
luying@iss.nus.sg

Abstract
To support future digital library systems which draw information from different sources on the internet, we have to provide integrated queries to pre-existing database servers which contain structured, semi-structured and unstructured data. In this paper, we specifically examine the problem of querying both existing structured relational databases and bibliographic databases. By adopting the well-accepted Z39.50 standard protocol to access bibliographic databases in different legacy library systems, we have developed an extended SQL model, known as HarpSQL, to support integrated queries to both SQL databases and bibliographic databases. Using HarpSQL, one can not only query bibliographic databases in an SQL manner, but also perform join(s) between SQL and bibliographic databases. A distributed query processor supporting HarpSQL queries has been developed. We will present our query processing strategy that is based on client-server interaction model between the distributed query processor and the various remote database servers.

Keywords Digital libraries, internet databases, interoperable databases

1 Introduction
1.1 Motivation
As increasing number of databases are being made available on the internet, it is now possible to build a wide variety of global applications which make use of data from different sources. An example of such global applications is a digital library system. In [4], a digital library is defined to be a machine readable collection of information together with a set of tools that helps users to find specific information. Here, the collection of information is no longer restricted to the information owned by a single public library. Digital library systems are expected to interoperate with a wide range of information servers managed by different information providers. The information provided by these servers includes:

- Bibliographic data:
Bibliographic data (sometimes known as the library catalog) exist in every public library. The creation of bibliographic records for library material is usually done by professional catalogers. To look for any library material, one always has to begin with searching the library’s bibliographic database. Hence, bibliographic data represent an important class of information provided by the existing public libraries. Lately, new forms of bibliographic data have emerged due to the need to index publications available on the internet. For example, the Unified Computer Science Technical Report Index (UCSTRI) maintained by University of Indiana [19] and other WWW bibliographies [2, 7] have been constructed for computer science related bibliographic information. Nevertheless, these new index servers may not adopt a common query interface to their bibliographic data and they also do not capture the large bulk of bibliographic data maintained by the public libraries.

- Structured data:
Structured data have traditionally been used to store business and organization information. As SQL database systems become inexpensive, they are becoming widely used. Since modern SQL database systems can also be used...
to store text data and they provide sophisticated text query features, we expect many of the SQL databases will be used as components of digital libraries and will be used to stored information related to digital libraries. For example, SQL databases may be used to stored information about inter-library loan requests, and books that have been borrowed or reserved.

- Document data:
Document data on the internet can exist in a variety of formats. Some of them are totally unstructured, e.g. plain text files. Others may be semi-structured. They can be represented by some mark-up languages such as SGML[6], HTML[5], etc. At present, the most popular way to obtain remote document files is through a web browser. A large number of document files can also be obtained from ftp (file transfer protocol) and WAIS[8] (wide area information servers) sites.

In this paper, we address the important query processing problem which involves both existing bibliographic databases owned by the public libraries and SQL databases containing structured data. For example, to perform interlibrary loan, a library user has to first register his request(s) with his affiliated library. Suppose interlibrary loan requests are stored in a SQL database. After the librarian has approved a number of registered requests, a query process that attempts to locate the requested books in neighboring public libraries will be carried out. In this process, it would be useful to provide integrated queries to both bibliographic databases and SQL databases. Another query example that involves both types of databases will be given in Section 3.

By adopting a SQL-like query language and the Z39.50 information retrieval protocol[13] to access the existing bibliographic databases, we examine how queries to these legacy bibliographic databases can be evaluated. We further study the strategy of performing joins between SQL tables and bibliographic data. We have also developed a distributed query processor that incorporates the tuple substitution join strategy.

1.2 Distributed Query Processing Issues for Structured and Bibliographic Databases

Unlike the traditional distributed query processing problem, processing integrated queries to structured and bibliographic databases has to consider a number of issues:

- Modeling of existing bibliographic databases:
2.1 Multidatabase Query Processing

Similar to multidatabase systems, digital library systems have to accommodate different types of autonomous and heterogeneous databases. However, most multidatabase research has focused on querying distributed structured databases only. Multidatabase query processing can also be very complicated when the semantic conflicts between participating databases have to be resolved.

In [17], a multidatabase query processing strategy has been proposed. It, however, did not consider semantic conflict resolution in the query processing strategy. To resolve inter-database conflicts, Lim etc. has proposed new integration operations in [10]. In [11], a new object-oriented data model known as DIOM has been designed for the Diorama multidatabase project.

2.2 Distributed Query Processing for Semi-structured Data

To model and query all kinds of semi-structured data, Quass etc. have proposed a flexible data model and query language known as OEM and LOREL respectively[16, 14]. Blake etc. have extended SQL to query semi-structured data and their meta-description. Unlike other types of semi-structured data, bibliographic data in the public libraries are stored as MARC records. Hence, we are able to model the bibliographic databases as relations and to extend SQL to query them.

Although distributed query processing problem has been well studied in the domain of relational databases, there is very little research effort in processing distributed queries which involve both structured and bibliographic databases. In [2], several join techniques have been proposed for queries which involves an external text data manager loosely coupled with a relational database system. These techniques include (a) naive tuple substitution, (b) relational text processing, (c) semijoin and (d) probing.

The naive tuple substitution technique requires a join between relation and text to be translated into a set of selection queries to the text database. This is done by evaluating the relational query followed by substituting relational attribute in the join predicate by its actual column values. This technique is usually undesirable because a large overhead will incur when numerous selection queries are sent to the text database. The relational text processing technique assumes that the relational database system can handle the join predicates between relational attributes and text attributes. This assumption however, does not hold in our context because the extended predicates and functions in our integrated queries cannot be handled by ordinary relational database systems. In [2], the proposed semijoin technique is actually a variant of tuple substitution. Semijoin reduces the overhead of...
tuple substitution by combining all selection queries
generated by tuple substitution into one selection
query. The probing technique, designed to work
with either naive tuple substitution or rela-
tional text processing, further improves the two
techniques by not sending queries that return empty
results to the text system.

In this paper, since our queries involve multiple
external SQL and bibliographic databases, the dis-
tributed query processing problem becomes more
complex than that examined by [2]. We have adopt-
ed a tuple substitution approach similar to semi-
join to evaluate subqueries on bibliographic databases.
Probing method is not chosen because we currently
do not maintain the statistics the probing method
requires for the external databases.

3 HarpSQL Query Language
To enable digital library users and application de-
velopers to query existing SQL and bibliographic
data, we have extended the SQL language in a
number of ways and called it HarpSQL[12]. The
unique features of HarpSQL include:\footnote{Due to space constraint, we only list the major exten-
sions that are relevant to our discussion.}

- **Foreign SQL and bibliographic tables**
  In HarpSQL, a table (say CourseTB) from a
  remote SQL database (say RefDB) can be im-
  ported as a foreign SQL table (named as
  \texttt{CourseTB0RefDB}). Unlike SQL databases, re-
  mote bibliographic databases do not contain
  member tables. Hence, each remote bibli-
  ographic database is imported as a foreign Bib-
  liographic table (or BIB table). Each im-
  ported BIB table is named BibTBQ<Library
  name> where <Library name> is the public
  library that provides the BIB table.

- **MARCString data type**
  Bibliographic data found in the public libraries
  are mostly formatted based on the MARC stan-
  dard. To model the MARC fields, we have de-
  fined a new data type called \texttt{MARCString}.
  A MARCString value models multiple MARC
  fields sharing a common tag number in a MARC
  record. These MARC fields form the elements
  of the MARCString value. Each element may
  consist of multiple subelements modeling the
  subfields in the MARC fields. Hence, an im-
  ported BIB table consists of multiple MARC-
  String attributes each with a unique tag num-
  ber and attribute name \texttt{MAttr<tag number>}. For
  example, the \texttt{MAttr850} attribute value of
  the record given in Figure 2 is:

\begin{verbatim}
(650, ('$a Business' ' $a Data processing')
('a Information storage and retrieval systems')
\end{verbatim}

![Figure 3: Schema of RefDB](image)

\begin{verbatim}
RefTB Title Author Course
RefId | Title | Author | Course
\end{verbatim}

\begin{verbatim}
CourseTB CourseId Cname Year Lecturer
\end{verbatim}

\begin{verbatim}
CourseId Cname Year Lecturer
\end{verbatim}

\begin{verbatim}
('a Business')
('a Information technology')
('a Local area networks(Computer networks)')
\end{verbatim}

- **Virtual bibliographic tables**
  To support broadcasting of queries to multi-
  ple bibliographic databases, HarpSQL allows a
  virtual bibliographic (BIB) table to be
  defined upon a number of import BIB tables
  which are also known as the members BIB
  tables. Apart from having the same MARC-
  String attributes found in any BIB table, every
  virtual BIB table contains an extra location
  attribute to indicate where its records come
  from.

- **Contain predicate and Extract function**
  With the new data type \texttt{MARCString}, a new
  predicate called \texttt{Contain} has been defined to
  apply different kinds of selection criteria on
  MARCString values, and to allow BIB tables
  to be joined with SQL tables by comparing
  the MARCString attributes with the charac-
  ter string attributes in the SQL tables. Un-
  like the usual regular expression predicates,
  the \texttt{Contain} predicate caters for a wide vari-
  ety of string comparison methods by support-
  ing different search modes for different bibli-
  ographic elements (see [12] for details about
  search modes). \texttt{Extract} function, on the other
  hand, allows us to extract sub-elements from a
  MARCString value by supplying the subtags.

Example: Let \texttt{BibTB0NTU} and \texttt{BibTB0NUS} be
two BIB tables imported from the NTU\footnote{NTU is an abbreviation of Nanyang Technological
University.} library and NUS\footnote{NUS is an abbreviation of National University of
Singapore.} library. Let \texttt{Course0RefDB} and
\texttt{RefTB0RefDB} be two SQL tables imported from
\texttt{RefDB}, a SQL database containing some course in-
formation. \texttt{RefTB0RefDB} contains information about
reference books adopted by different courses.
\texttt{Course0RefDB} contains information about the courses
to be taken by computer engineering students. Their
attributes are shown in Figure 3.

In the following, we show some query examples\footnote{To simplify our explanation, some parameters to be used
in \texttt{Extract} and \texttt{Contain} are not shown.}
demonstrating the HarpSQL features.

**Example (Q1):** Retrieve the titles and au-
thors of books with titles containing 'distributed
\begin{verbatim}
\end{verbatim}
database' from the NTU library.

```sql
SELECT Extract(MAttr245,'$a'),
       Extract(MAttr100,'$a') FROM BibTBOINTU
WHERE Contain(MAttr245,'distributed database',<ANY.POSITION,IS.PHRASE>)
```

In the above HarpSQL query, MAttr245 and MAttr100 are the MARCString attributes containing the title and author information in their subelements with subtag $a. The search mode <ANY.POSITION,IS.PHRASE> in the Contain predicate indicates that only those titles containing 'distributed database' as a phrase are wanted. The Extract functions are used to obtain title and author text from the MARCString attributes MAttr245 and MAttr100 respectively.

Example (Q2): Retrieve the course titles, call numbers, titles, authors and locations of reference books used by courses held in the academic year 95/96 from NTU and NUS libraries.

```sql
SELECT c.Cname, Extract(a.MAttr092,'$a'),
       Extract(a.MAttr245,'$a'),
       Extract(a.MAttr100,'$a'), a.location
FROM NTUandNUSLib a, RefTBORefDB b,
    CourseTB@RefDB c
WHERE c.Year = '95/96' AND
    b.Course = c.CourseId AND
    Contain(a.MAttr245,b.Title,<FIRST.IN.SUBFIELD,IS.PHRASE>) AND
    Contain(a.MAttr100,b.Author,<NULL,IS.NAME>)
```

The MARCString attribute MAttr092 contains the call number information. The above query specifies a join between two SQL tables and a virtual BIB table NTUandNUSLib defined on the BIB tables imported from NTU and NUS libraries. Note that the Contain predicates have been used to join RefTB with NTUandNUSLib.

## 4 Distributed Query Processing Architecture

Our distributed query processor consists of mainly a query manager and a set of query agents as shown in Figure 4. Query manager is the core of the distributed query processor. It coordinates the entire query processing by interacting with its query agents. Given a HarpSQL query from a digital library application, the query manager first parses it into a query graph that is later decomposed into a number of subqueries to be processed by the query agents. Having collected all the subquery results, the query manager combines them together and returns the final query result to the digital library application. Since not all operations of a query can always be performed by the query agents, the query executor has to handle some operations at the query manager site. Hence, a HarpSQL server is needed to supplement the query executor with the capabilities to store and process intermediate results.

![Figure 4: Architecture of the Distributed Query Processor](image)

Figure 4 also depicts the remote SQL and 239.50 servers managing existing structured data and bibliographic data respectively. The SQL and 239.50 query agents act as wrappers that support subqueries to remote SQL and 239.50 servers which are members of the integrated digital library environment. A query agent receives subqueries from the query manager, sends them to its remote database server for processing and returns the result to the query manager. The interaction between the query agents and their remote servers are governed by the specific remote access protocols supported by the servers. By using the query agents, the query manager is able to execute queries without knowing much about the complex protocols and query interfaces adopted by the remote servers. Furthermore, the query agents are designed to process their subqueries concurrently, thus shortening the query response time.

## 5 HarpSQL Query Processing Strategy

In this section, we describe the query processing strategy adopted by our HarpSQL distributed query processor which has been developed based on the architecture given in Section 4. Although query optimization is not the prime focus of this research, our processing strategy has been designed to reduce the subquery results by performing selection and projection as early as possible and by avoiding cartesian products in the subqueries to be evalu-
ated by the external servers. By reducing the subquery results, we are able to minimize the overhead of shipping data from the external servers to the distributed query processor. Upon receiving the subquery results, the Harps&L server will combine them together by performing some inter-database joins or cartesian products.

5.1 Restricting Bibliographic Queries using Tuple Substitution

As Z39.50 disallows bibliographic queries that do not carry any selection predicate\(^6\), our query processing strategy requires all BIB tables involved in HarpSQL queries to be restricted by either selection or join with other SQL tables. For those BIB tables that are only restricted by join, we can derive the subqueries to their Z39.50 servers by performing tuple substitution\(^7\). In tuple substitution, a join predicate used in the join between a BIB table and a SQL table is transformed into a disjunctive set of selection predicates by first evaluating the SQL table, followed by instantiating the SQL attribute in the join predicate by the corresponding attribute values in the SQL subquery result.

For example, to process the query (Q3) below, we first evaluate the SQL subquery to obtain the various reference title values from \text{RefTBQRefDB}.

**Example (Q3):**

```sql
SELECT Extract(a.MAttr092,'$a'), a.HAttr245
FROM BibTBQNTU a, RefTBQRefDB b,
WHERE Contain(a.MAttr245,b.Title,
<FIRST_IN_SUBFIELD,IS_PHRASE>)
```

Suppose the titles returned are 'Digital Design', 'Computer Networks', ... By tuple substitution, we obtain the following selection subquery for \text{BibTBQNTU}:

```sql
SELECT *
FROM BibTBQNTU
WHERE Contain(MAttr245,'Digital Design',
<FIRST_IN_SUBFIELD,IS_PHRASE>) OR
```

5.2 Distributed Query Processing Steps

To process a HarpSQL query, we first represent it using a query graph[20]. In a query graph, each node represents a SQL table, BIB table or virtual BIB table. An edge between a pair of nodes represents a join. For example, the query graph representing Q2 in Section 3 is shown in Figure 5. Given a query graph, the query processing steps

\(^6\)This prevents huge amount of bibliographic data to be shipped across sites.

\(^7\)This is similar to the semijoin technique mentioned in [2].

**Figure 5: A Query Graph Example**

For each \(N \in V\) {
  
  ```
  mark \(N\) as UNVISITED
  \(SId = 1\)
  ```

  ```
  for each unvisited \(N \in V\) {
    
    ```
    SQLSubgraph\(N\).V = 4
    SQLSubgraph\(N\).E = q5
    ```
    
    ```
    if \((N.type == SQLTable)\) {
      DepthFirstSearch(N)
      SId ++
    }
    ```

  }

  ```
  DepthFirstSearch(A) {
    SQLSubgraph\(A\).V \cup = \{A\}
    mark A as VISITED
    ```
    ```
    for each edge \((A, B) \in E\) {
      ```
      ```
      ```
      if B is UNVISITED and B is a SQL table at the same site as A {
        SQLSubgraph\(A\).E \cup = \{(A, B)\}
      ```
    }
  ```
  ```

  ```
  ```

  ```

- **Step 1: SQL subgraph extraction**
  When a query graph consists of SQL table(s), we first derive the subqueries to these tables by extracting SQL subgraphs from the query graph. A SQL subgraph is a connected subgraph of query graph consisting of SQL tables that belong to the same SQL database.

  Let the query graph be represented by \((V, E)\) where \(V\) and \(E\) denote the set of nodes and edges respectively. The following algorithm derives a set of SQL subgraphs denoted by \{SQLSubgraph\(1\), SQLSubgraph\(2\), ...\}.

  for each \(N \in V\) {
    ```
    mark \(N\) as UNVISITED
    ```
    ```
    SId = 1
    ```
    ```
  }

  for each unvisited \(N \in V\) {
    ```
    SQLSubgraph\(N\).V = 4
    SQLSubgraph\(N\).E = q5
    ```
    ```
    if \((N.type == SQLTable)\) {
      DepthFirstSearch(N)
      SId ++
    }
  }

  ```

  DepthFirstSearch(A) {
    SQLSubgraph\(A\).V \cup = \{A\}
    mark A as VISITED
    ```
    ```
    for each edge \((A, B) \in E\) {
      ```
      ```
      ```
      if B is UNVISITED and B is a SQL table at the same site as A {
        SQLSubgraph\(A\).E \cup = \{(A, B)\}
      ```
    }
  ```
  ```

- **Step 2: Processing SQL subqueries**
  Once the SQL subgraphs are extracted, we generate a SQL subquery for each subgraph. All these SQL subqueries are submitted to the SQL query agents created for the target SQL database servers and are evaluated by the servers concurrently. Typically, the SQL subqueries
involve select, project and intra-database join operations. A temporary table for each subquery result is created at the HarpSQL server when the subquery result is returned by the query agent.

Step 3: Processing bibliographic subqueries
In this step, we derive and evaluate the subqueries against the BIB tables. These subqueries can be obtained in two ways as described below:

Case (a): If a BIB table (or virtual BIB table) node in the query graph is restricted by some selection predicate(s), a bibliographic subquery against the BIB table (or virtual BIB table) with the selection predicate(s) is derived.

Case (b): If a BIB table (or virtual BIB table) node in the query graph is not restricted by any selection predicate(s), we have to derive the bibliographic subquery by performing tuple substitution. In tuple substitution, the subquery result of a SQL subgraph that is linked to the BIB table (or virtual BIB table) is chosen to convert a join predicate between the SQL subgraph and BIB table (or virtual BIB table) into a disjunction of selection predicates.

For each subquery against a BIB table, we create a 239.50 query agent to process it. The subquery result returned by the query agent is stored as a temporary table at the HarpSQL server with the necessary attribute projection. In both case (a) and (b), a subquery against a virtual BIB table will be further replicated into subqueries against its different member BIB tables. Multiple 239.50 query agents, each corresponding to a member BIB table, will be created to process these subqueries. The results of all these subqueries are unioned and stored in a temporary table in the HarpSQL server. Finally, a query that joins the temporary tables is evaluated by the HarpSQL server to obtain the final query result.

Step 4: Final result generation
A final query that joins all the temporary tables at the HarpSQL server is generated. Apart from the final attributes to be projected, the final query may also involve Contain predicates and Extract functions.

When the HarpSQL queries to be processed involve only SQL tables, only steps 1, 2 and 4 are required. On the other hand, if a HarpSQL query involves only a BIB table or virtual BIB table, we only need to perform steps 3 and 4.

5.3 Query Processing Example
Figures 6 shows the query processing steps for our query example Q2. From the query graph (see Figure 5), we extract a SQL subgraph which is translated into a SQL subquery to be executed by a SQL query agent. The SQL subgraph is shown in Figure 6(a). From the subgraph, we generate a SQL subquery and send it to a SQL query agent as shown in Figure 6(b). A temporary table T1 is created at the HarpSQL server for the subquery result. Subsequently, we substitute the Title attribute in the Contain predicate by the corresponding attribute values in the previous SQL subquery result T1. A substituted BIB subquery is created and is submitted to the two BIB query agents for the NTU and NUS libraries as shown in Figure 6(c). The results from all the BIB query agents are unioned and stored in a temporary table T2 by the HarpSQL server. Finally, a query that joins T1 and T2 is evaluated by the HarpSQL server to obtain the final query result as shown in Figure 6(d).

6 Implementation Issues
As part of our research work, we have developed a distributed query processor that can handle HarpSQL queries over a collection of SQL and 239.50 servers. Within the distributed query processor, the query manager and agents are implemented as separate processes. Message queues have been used for communication between the query manager and agent processes.

Since the query manager is responsible for storing and processing intermediate results collected from different remote servers, we need a HarpSQL server which can handle both SQL and bibliographic data on behalf of the query manager. In the following subsection, we describe how we realize the HarpSQL server by extending the POSTGRES database system.

6.1 HarpSQL Server Implementation
To play a role in processing distributed HarpSQL queries, the HarpSQL server supplementing the query manager must support the following features:

- Basic SQL data types and the MARCString data type
- Contain() predicate
Extracted SQL Subgraph:

```
CourseTB@RefDB
  b.COURSE = c.CourseId
  c.Year = '95f96'
```

(a) SQL Subgraph Extraction

Generated SQL Subquery:

```
Select c.Cname, b.Title, b.Author
From RefTB b, CourseTB c
Where bOURSE = c.COURSE Id And
  c.Year = '95f96'
```

Subquery Result is stored as T1 in the HarpSQL server

(b) Processing SQL Subquery

Generated Bibliographic Query using Tuple Substitution:

Selection predicates:

- Contain(MAttr245,"Digital Design",...) Or
- Contain(MAtt6&45,"Computer Networks",...) Or

Subquery submitted to NTU and NUS servers are unioned and stored as T2 in the HarpSQL server

(c) Processing Bibliographic Subquery

Generated Final Query:

```
Select T1.Cname, Extract(T2.MAttrO92,'$a'), Extract(T2.MAttr245,'$a'), Extract(T2.MAttr100,'$a')
From T1, T2
Where Contain(T2.MAttr100,T1.Author,<NULL,IS-NAM&
```

(d) Final Result Generation

Figure 6: Query Processing Steps for Q3

- **Extract() function**

Instead of building the HarpSQL server from scratch, we base our implementation on the POSTGRES database system [18]. One key difference between POSTGRES and standard relational systems is that POSTGRES captures extra information in its catalog which allows its processing and storage capabilities to be extended. This includes not only information about tables and fields, but also information about types, functions, access methods, and etc. These information can be modified by the user, and POSTGRES carries out its internal operation based on these information. The query language of POSTGRES is known as POSTQUEL. POSTGRES can also incorporate pre-compiled user-written code into its query processing through dynamic loading. In other words, the user can create an object file (e.g., a compiled .o file or shared library) that implements new types and function in POSTGRES. A detailed description can be found in [15].

Figure 7 shows the architecture of the HarpSQL server. Our HarpSQL server is developed by augmenting POSTGRES with the MARCString data type and its extended predicates and functions.

**MARCString data type**

In order to support the MARCString data type, a C data structure is defined and is used by the POSTGRES database system as an internal representation of a MARCString value. The MARCString data type can be incorporated into POSTGRES by the following POSTQUEL statements:

1. define function marcString.in
   (language = "c",
    returntype = MARCString)
   arg is (any)
   as "/home/harp/lqahs/marc.so"

2. define function marcString.out
   (language = "c",
    returntype = any)
   arg is (any)
   as "/home/harp/lqahs/marc.so"

3. define type MARCString
   (intenlength = 2088,
    input = marcString.in,
    output = marcString.out)

In statements (1) and (2), the input function marcString.in and output function marcString.out are defined. They are used to determine how the MARCString appears in strings (for input by the user and output to the user). The MARCString data type is specified in statement (3). The MARCString data structure and the actual implementation of the input and output functions are included in the object file marc.so.

**Extended predicates and functions**

The following POSTQUEL statements are used to incorporate Contain predicate and the Extract function into POSTGRES:

4. define function contain
   (language="c",
    returntype = bool)
   arg is (MARCString,text,int4,int4)
   as "/home/harp/lqahs/marc.so"

5. define function extract
   (language = "c",
    returntype=text)
   arg is (MARCString,text,int4,int4)
   as "/home/harp/lqahs/marc.so"

The actual implementation of Contain and Extract is included in the object file marc.so.
7 Conclusions

In this paper, we propose the use of HarpSQL, an extension of SQL, to formulate integrated queries to legacy bibliographic databases and SQL databases. HarpSQL supports new data type, predicate and function required for representing and manipulating the MARC formatted bibliographic data. By accommodating the MARC formatted data, and by adopting the Z39.50 protocol standard to access the bibliographic databases in the public libraries, we achieve interoperability while not sacrificing the local autonomy of the existing library systems. HarpSQL also supports joins between SQL and bibliographic data.

To process HarpSQL queries over SQL and bibliographic databases at different locations, we have designed and implemented a distributed query processor which adopts some heuristics to reduce communication costs during query processing. To handle inter-database joins including joins between SQL and bibliographic data, we implemented a HarpSQL server which provides the query processing capabilities to the query manager. Moreover, we have also implemented a user-friendly graphical query frontend for users to formulate their HarpSQL queries.

A digital library system typically consists of three layers of software, namely the digital library applications, digital library services, and information servers as shown in Figure 8. The work presented in this paper represents an effort in the digital library service layer. Our distributed query processing technique therefore renders an important step towards advanced query support for future digital library applications.

We are currently extending our work in several directions. First, we are considering the use of cost based optimization techniques in processing the HarpSQL queries. Second, we plan to extend the HarpSQL to query other forms of data, e.g. Web pages, since the latter represents a fast growing source of information on the internet. Finally, we are attempting to build some advanced digital library applications, e.g. interlibrary loan, using HarpSQL and our distributed query processor.

7.1 Acknowledgements

We thank Zhiliang Wang from the Information Technology Institute (ITI), National Computer Board for helpful discussions on the implementation issues. We are also grateful to our library staff and computer center colleagues for providing valuable information about the bibliographic databases maintained by the Nanyang Technological University Library.

References


