

4-1997

# Distributed query processing for structured and bibliographic databases


Ee Peng LIM

Singapore Management University, eplim@smu.edu.sg

Ying LU

**DOI:** [https://doi.org/10.1142/9789812819536\\_0046](https://doi.org/10.1142/9789812819536_0046)

Follow this and additional works at: [https://ink.library.smu.edu.sg/sis\\_research](https://ink.library.smu.edu.sg/sis_research)

 Part of the [Databases and Information Systems Commons](#), and the [Numerical Analysis and Scientific Computing Commons](#)

---

## Citation

LIM, Ee Peng and LU, Ying. Distributed query processing for structured and bibliographic databases. (1997). *Database Systems for Advanced Applications '97: Proceedings of the Fifth International Conference on Database Systems for Advanced Applications: Melbourne, April 1-4, 1997*. 441-450. Research Collection School Of Information Systems.

**Available at:** [https://ink.library.smu.edu.sg/sis\\_research/920](https://ink.library.smu.edu.sg/sis_research/920)

This Conference Proceeding Article is brought to you for free and open access by the School of Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email [libIR@smu.edu.sg](mailto:libIR@smu.edu.sg).

# 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

Proceedings of the Fifth International Conference on Database Systems for Advanced Applications, Melbourne, Australia, April 1-4, 1997.

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[9, 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 store information related to digital libraries. For example, SQL databases may be used to store 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:**

Tag No.	Field name
001	Control number
020	ISBN number
040	Cataloging source
092	Call number
100	Main entry - personal name
110	Main entry - corporate name
111	Main entry - conference or meeting
245	Title statement
250	Edition statement
260	Publisher
600	Subject added entry - personal name
610	Subject added entry - corporate name
650	Subject added entry - topical heading

Figure 1: Selected MARC Fields and Tags

Most of the existing bibliographic databases contain bibliographic records represented in the MARC<sup>1</sup> format[3]. Each MARC record consists of multiple fields representing bibliographic elements, e.g. title, author, subject, ISBN, etc. These bibliographic elements are identified by unique tag numbers (defined by the MARC standard). A selected set of MARC fields and their tags are shown in Figure 1. A bibliographic record formatted in MARC is shown in Figure 2. Some bibliographic elements may occur more than once in the same record and each occurrence contains a different value (e.g. the example MARC record has multiple fields with the tag 650). Furthermore, a field may be composed by one or more subfields each carrying a specific meaning and a subtag (e.g. \$a). In this paper, we will present an extended SQL model known as HarpSQL to query these existing bibliographic data.

- **Query capabilities of remote access protocols:**

To ensure that our distributed query processing strategy is applicable to the present and future bibliographic databases, we have adopted the Z39.50 protocol to query these databases[13]. Z39.50 is an application layer information retrieval protocol drafted by ANSI/NISO. It has been widely used to support remote accesses to the bibliographic databases maintained by public libraries. At present, the queries supported by most Z39.50 servers are restricted to boolean searches (or selection queries) which consist of predicates on the MARC fields connected by boolean operators (AND, OR, NOT). Projection and join operations are not supported. In other words, our distributed query processing strategy has to observe the query restriction imposed by Z39.50. For example, our query processor has to ensure that only selection queries are submitted to Z39.50 servers. Moreover, the query processor must support

<sup>1</sup>MARC is the abbreviation of MACHine-Readable Cataloging.

#### Bibliographic record:

Senn, James A. Information technology in business: principles, practices, and opportunities. Annotated instructor's ed. Englewood Cliffs, N.J.: Prentice Hall, c1995.

#### Corresponding MARC record:

```
020 $a 0134849086 (Instructor's ed.)
020 $a 0134843045 (Student ed.)
040 $a DLC $c DLC $d DLC
100 $a Senn, James A.
245 $a Information technology in business :
    $b principles, practices, and opportunities
    $c James A. Senn.
250 $a Annotated instructor's ed.
260 $a Englewood Cliffs, N.J. : $b Prentice Hall,
    $c c1995.
650 $a Business $x Data processing.
650 $a Information storage and retrieval systems
    $x Business.
650 $a Information technology.
650 $a Local area networks (Computer networks)
```

Figure 2: A Bibliographic Record Example Formatted in MARC

join and projection operations which are not part of Z39.50 query support.

- **Merging different kinds of data:**

Apart from retrieving data from remote SQL and bibliographic databases, our query processor has to be able to merge the retrieved SQL and bibliographic data. In our approach, the distributed query processor supports extended predicates to be used in joining the two kinds of data.

### 1.3 Paper Outline

The rest of this paper is organized as follows. In Section 2, we discuss some related work. Section 3 describes an extended query language (HarpSQL) for writing integrated queries to bibliographic and SQL databases. Section 4 presents our distributed query processing architecture. Processing HarpSQL queries will be given in Section 5. Implementation of our distributed query processor is described in Section 6. Conclusions are given in Section 7.

## 2 Related Work

As bibliographic data is semi-structured, our research is related to some ongoing work in modeling and querying semi-structured data [16, 1, 14]. On the other hand, since we are dealing with distributed heterogeneous databases, our work is also related to the current research in multidatabase query processing [17, 10, 11]. In the following, we will survey these two related research areas.

### 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 together with either naive tuple substitution or relational 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 distributed query processing problem becomes more complex than that examined by [2]. We have adopted a tuple substitution approach similar to semijoin 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 developers 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<sup>2</sup>:

- **Foreign SQL and bibliographic tables**  
In HarpSQL, a table (say **CourseTB**) from a remote SQL database (say **RefDB**) can be imported as a foreign SQL table (named as **CourseTB@RefDB**). Unlike SQL databases, remote bibliographic databases do not contain member tables. Hence, each remote bibliographic database is imported as a foreign **Bibliographic table** (or **BIB table**). Each imported BIB table is named **BibTB@<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 standard. To model the MARC fields, we have defined a new data type called **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 imported BIB table consists of multiple **MARCString** attributes each with a unique tag number and attribute name **MAttr<tag\_number>**. For example, the **MAttr650** attribute value of the record given in Figure 2 is:

```
(650, ('$a Business' '$x Data processing')
('$a Information storage and retrieval systems')
```

<sup>2</sup>Due to space constraint, we only list the major extensions that are relevant to our discussion.

RefTB	RefId	Title	Author	Course
CourseTB	CourseId	Cname	Year	Lecturer

Figure 3: Schema of **RefDB**

```
('x Business')
('$a Information technology')
('$a Local area networks(Computer networks)')
```

- **Virtual bibliographic tables**  
To support broadcasting of queries to multiple 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 MARCString 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 **MARCString**, a new predicate called **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 character string attributes in the SQL tables. Unlike the usual regular expression predicates, the **Contain** predicate caters for a wide variety of string comparison methods by supporting different search modes for different bibliographic elements (see [12] for details about search modes). **Extract** function, on the other hand, allows us to extract sub-elements from a **MARCString** value by supplying the subtags.

**Example:** Let **BibTB@NTU** and **BibTB@NUS** be two BIB tables imported from the NTU<sup>3</sup> library and NUS<sup>4</sup> library. Let **Course@RefDB** and **RefTB@RefDB** be two SQL tables imported from **RefDB**, a SQL database containing some course information. **RefTB@RefDB** contains information about reference books adopted by different courses. **Course@RefDB** 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<sup>5</sup> demonstrating the HarpSQL features.

**Example (Q1):** Retrieve the titles and authors of books with titles containing 'distributed

<sup>3</sup>NTU is an abbreviation of Nanyang Technological University.

<sup>4</sup>NUS is an abbreviation of National University of Singapore.

<sup>5</sup>To simplify our explanation, some parameters to be used in **Extract** and **Contain** are not shown.

database' from the NTU library.

```
SELECT Extract(MAttr245,'$a'),
       Extract(MAttr100,'$a') FROM BibTB@NTU
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.

```
SELECT c.Cname, Extract(a.MAttr092,'$a'),
       Extract(a.MAttr245,'$a'),
       Extract(a.MAttr100,'$a'), a.location
FROM NTUandNUSLib a, RefTB@RefDB 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

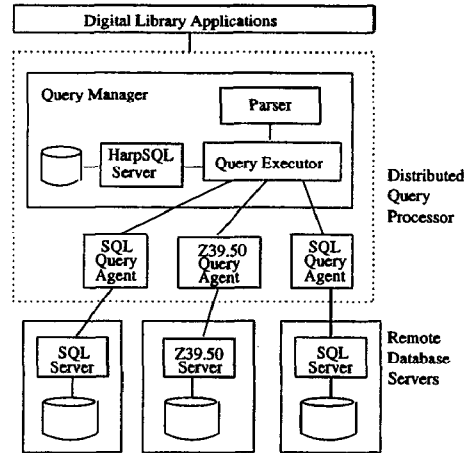


Figure 4: Architecture of the Distributed Query Processor

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 also depicts the remote SQL and Z39.50 servers managing existing structured data and bibliographic data respectively. The SQL and Z39.50 query agents act as wrappers that support subqueries to remote SQL and Z39.50 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 HarpSQL 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<sup>6</sup>, 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**<sup>7</sup>. 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 RefTB@RefDB.

### Example (Q3):

```
SELECT Extract(a.MAttr092, '$a'), a.MAttr245
FROM BibTB@NTU a, RefTB@RefDB 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 BibTB@NTU:

```
SELECT *
FROM BibTB@NTU
WHERE Contain(MAttr245, 'Digital Design',
<FIRST_IN_SUBFIELD, IS_PHRASE>) OR
Contain(MAttr245, 'Computer Networks',
<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

<sup>6</sup>This prevents huge amount of bibliographic data to be shipped across sites.

<sup>7</sup>This is similar to the semijoin technique mentioned in [2].

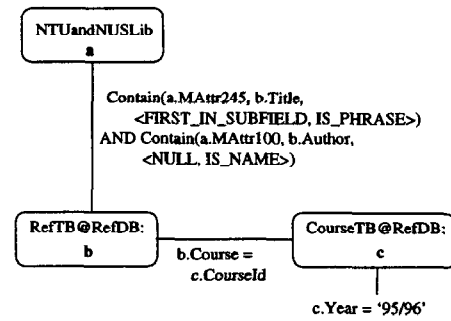


Figure 5: A Query Graph Example

performed by our distributed query processor are as follows:

- **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  $\{SQL\_Subgraph_1, SQL\_SubGraph_2, \dots\}$ .

```
for each  $N \in V$  {
  mark  $N$  as UNVISITED
   $Sid = 1$ 
}
for each unvisited  $N \in V$  {
   $SQL\_Subgraph_{Sid}.V = \phi$ 
   $SQL\_Subgraph_{Sid}.E = \phi$ 
  if ( $N.type == SQLTable$ ) {
     $DepthFirstSearch(N)$ 
     $Sid++$ 
  }
}
 $DepthFirstSearch(A)$  {
   $SQL\_Subgraph_{Sid}.V \cup = \{A\}$ 
  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  $A$  {
       $SQL\_Subgraph_{Sid}.E \cup = \{(A, B)\}$ 
       $DepthFirstSearch(B)$ 
    }
  }
}
```

- **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<sup>8</sup> 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 Z39.50 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 Z39.50 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 as a temporary table in the HarpSQL server with the necessary attribute projection. In the process of unioning the subquery results, the location attribute value is added to every record.

- **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 consist of (i) join(s) between tables from different SQL database servers, and (ii) join(s) between SQL table and BIB table (except the join used for tuple substitution). Apart from join operations, the final

<sup>8</sup> If there are multiple SQL subgraphs adjacent to the BIB table, we just choose one of them.

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

Figure 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 Z39.50 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



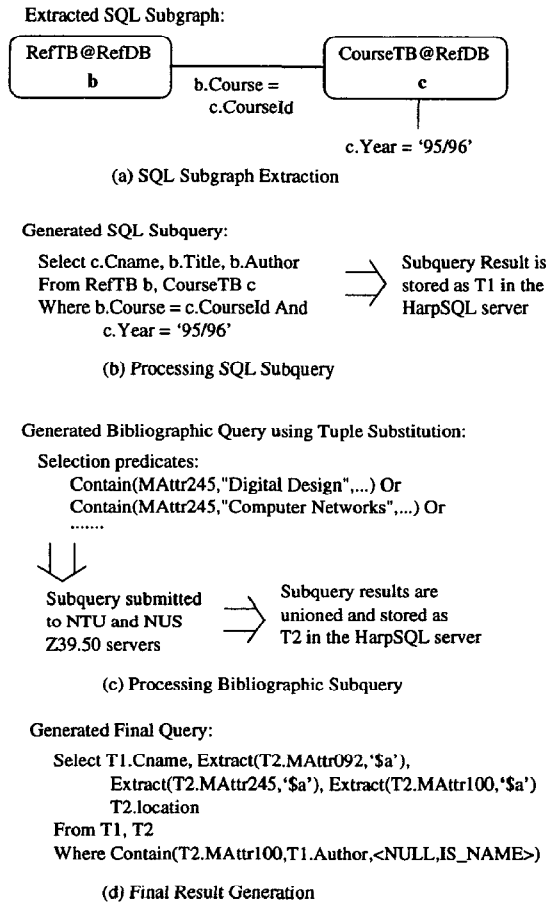


Figure 6: Query Processing Steps for Q3

- **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.

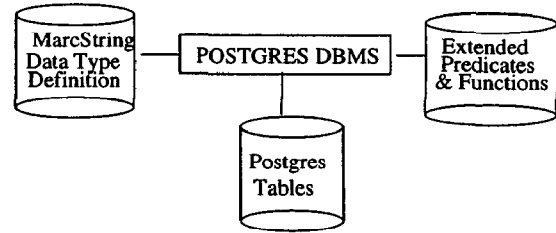


Figure 7: HarpSQL Server Architecture

- **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`  
`(internallength = 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`.

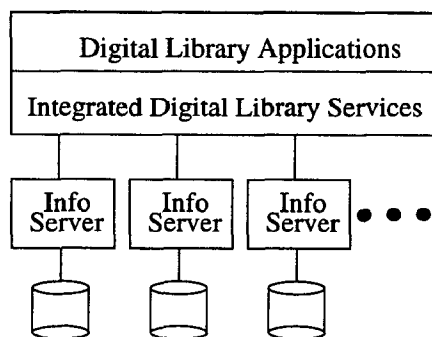


Figure 8: Integrated Digital Library Architecture

## 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

- [1] G.E. Blake, M.P. Consens, I.J. Davis, P. Kilpelainen, E. Kuikka, P.-A. Larson, T. Snider, and F.W. Tompa. *Text/Relational Database Management Systems: Overview and Proposed SQL Extensions Database Prototype*. Technical Report 95-25, UW Centre for the New OED and Text Research, University of Waterloo, 1995.
- [2] S. Chaudhuri, U. Dayal, and T.W. Yan. *Join Queries With Extended Text Sources: Execution and Optimization Techniques*. In *Proceedings of ACM SIGMOD Conference*, pages 410-422, San Jose, CA, 1995.
- [3] W. Crawford. *MARC for Library Use: Understanding the USMARC Formats*. Knowledge Industry Publications, Inc., 1984.
- [4] Henry M. Gladney, Nicholas J. Belkin, Zahid Ahmed, Edward A. Fox, Ron Ashany, and Maria Zemankova. *Digital Library: Gross Structure and Requirements (Report from a Workshop)*. Technical Report IBM Research Report RJ 9840, May 1994.
- [5] I. Graham. *The HTML Sourcebook*. John Wiley and Sons, 1995.
- [6] ISO. *International Standard 8879: Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML)*. First edition - 1986-10-15(ref. no. iso 8879-1986(e)) edition, 1986.
- [7] D. M. Jones. *The Hypertext Bibliography Project*. Technical report, Laboratory for Computer Science, MIT, 1996. <http://theory.lcs.mit.edu/dmjones/hbp/>
- [8] B. Kahle and A. Medlar. *An Information System for Corporate Users: Wide Area Information Servers*. *Conneziions - The Interoperability Report*, 5(11), Nov 1991.

- [9] M. Ley. DB&LP: A WWW Bibliography on Databases and Logic Programming. Technical report, Informatik Universitat Trier D-54286 Trier Germany, 1995.
- [10] E.-P. Lim, J. Srivastava, and S.-Y. Hwang. An Algebraic Transformation Framework for Multidatabase Queries. *Distributed and Parallel Database Journal*, 3(3), 1995.
- [11] L. Liu and C. Pu. Issues on Query Processing in Distributed and Interoperable Information Systems. In *Proceedings of the International Symposium on Cooperative Database Systems for Advanced Applications*, Kyoto, Japan, December 1996.
- [12] Ying Lu and Ee-Peng Lim. On Integrating Existing Bibliographic Databases and Structured Databases. In *IEEE Int'l Computer Software and Applications Conference*, August 1996.
- [13] National Information Standard Organization(NISO). ANSI Z39.50: Information Retrieval Service and Protocol, 1992.
- [14] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object Exchange Across Heterogeneous Information Sources. In *International Conf. on Data Engineering*, Taipei, March 1995.
- [15] The POSTGRES Group, Computer Science Div., Dept of EECS, University of California at Berkeley. *The POSTGRES User Manual*, 4.2 edition, 1994.
- [16] D. Quass, A. Rajaraman, Y. Sagiv, J.D. Ullman, and J. Widom. Querying Semi-structured Heterogeneous Information. In *4th International Conference on Deductive and Object-Oriented Databases*, Singapore, December 1995.
- [17] S. Salza, G. Barone, and T. Morzy. Distributed Query Optimization in Loosely Coupled Multidatabase Systems. In *International Conference on Database Theory*, Prague, 1994.
- [18] M. Stonebraker and L. Rowe. The Design of POSTGRES. In *Proceedings of 1986 ACM-SIGMOD Conference on Management of Data*, Washington, D.C., May 1986.
- [19] M.D. VanHeyningen. The Unified Computer Science Technical Report Index: Lessons in indexing diverse resources. In *Proceedings of the Second International WWW Conference*, Chicago, 1994.
- [20] E. Wong and K. Youssefi. Decomposition - A Strategy for Query Processing. *ACM Transaction on Database Systems*, 1(3), September 1976.