

A Java API for Global Querying and Updates for a System of Databases

Rajshekhar Sunderraman, Erdogan Dogdu, Praveen Madiraju and Laxmikanth Malladi

Department of Computer Science

Georgia State University

Atlanta, GA 30302

{raj,edogdu,cscpnmx}@cs.gsu.edu and csclnmx@tinman.cs.gsu.edu

ABSTRACT

In this paper, we present the design of system of databases (SyDb). We also give the design and implementation of a Java API for global querying and updates on the SyDb. The databases may be heterogeneous. The API allows for queries and updates that have global references to schema elements of multiple databases to be executed in a seamless manner. The API can be used to develop collaborative applications that need access to several independent databases on the network. One such collaborative application, called the Calendar application, is illustrated in the paper. In this application each individual keeps their schedule information in their personal database. The users can schedule meetings with others, view others schedules, cancel meetings, etc. We implement the API using direct JDBC connections to databases.

Categories and Subject Descriptors

H.2.4 [Database Management]: Systems – *distributed databases, query Processing.*

Keywords

Multidatabases, Global Queries/Updates, System of Databases, Distributed Information Retrieval

1. INTRODUCTION

A system of databases (SyDb) refers to a collaborating set of heterogeneous data resources. Global querying refers to the problem of information retrieval from heterogeneous and distributed sources. Global updates allow the user to store data on to a set of heterogeneous and distributed sources. The significant problems related to Global queries and updates are: (i) global schema mapping and integration [1,2,3,8,10], (ii) global query decomposition and optimization [10,13], and (iii) global constraint checking. [5,6,11,12]

However, the application developer should not be bogged down

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

43rd ACM Southeast Conference, March 18-20, 2005, Kennesaw, GA, USA. Copyright 2004 ACM 1-58113-000-0/00/0004...\$5.00.

with the above problems. Just as JDBC hides connection and retrieval details from the user, we need a framework/API that would hide all the complexities and empower the user with a ready to use package for Global queries and updates. To this effect, we propose a principled extension to the current SQL standard, SyDbQL. The SyDbQL would allow the user to specify global queries and global updates. We also set forth the design and implementation of SyDbQL Java API which would allow global queries and updates through a Java JDBC program.

Assumptions of the system: The assumptions of the system under consideration are: (i) a system of collaborating heterogeneous databases, (ii) schema integration and schema mapping of the individual component databases is resolved using the techniques such as [1,2,3,8,10], and (iii) an application which needs to query/update the component databases.

Contributions: Our main contributions in this paper are: (i) general framework for the SyDb with both global constraint checker and global query execution module, (ii) extensions to SQL, we call it SyDbQL i.e. SQL for a System of Databases, which allows global queries and updates, (iii) Java API for executing global queries and updates, and (iv) implementation of a collaborative calendar application using the proposed Java API.

The rest of the paper is organized as follows: In section 2, we give the system architecture and the design of the SyDb. The idea of global querying and syntax of SyDbQL is presented in section 3. A Java API for global query, update and its implementation using JDBC connections between databases is described in section 4. Section 5 introduces a collaborative application, calendar that uses the Java API and finally we describe related work and offer our conclusions in section 6.

2. DESIGN OF SyDb

The System of Databases (SyDb) architecture and the detailed steps involved in executing a global query/update is presented in Figure 1.

Step 1: Using the SyDbQL API, the user issues a global query/update.

Step 2: If the user issued a global query, the query is parsed, decomposed into a set of sub queries and sent to the component databases. The results obtained from the component databases are gathered, modified and finally the output is displayed. We discuss more about this in Section 4.

Step 3: If the user issued a global update, the update is input to the Constraint checker module. A general framework and an algorithm for the constraint checker module are given in [11,12]. Given an update statement *U* and the list of all global constraints *C*, the constraint checker module checks if any global constraints are violated without actually updating the database (compile time). In the current set up, global constraints can be stored in the temporary workspace.

Step 4: Constraint Checker generates sub constraint checks on to the component databases, gathers results and finally makes a decision if a constraint is violated. In case of non constraint violation, the global update statement is executed. The temporary workspace shown in Figure 1 is a local temporary workspace that a SyDbQL-based application can interact.

We broadly categorize data resources into three groups: (1) Type I: Relational databases that support remote clients, where data can be retrieved in Java by using JDBC drivers. (2) Type II: Relational Databases that do not support remote clients, where data can be retrieved by using either JDBC-ODBC bridge or by using JDBC drivers. (3) Type III: Remaining data resources, such as object-oriented databases, flat files, and XML data, where data can be retrieved using a wrapper that would convert the specific data format to relational tables and vice versa. The wrapper is data-source type dependent.

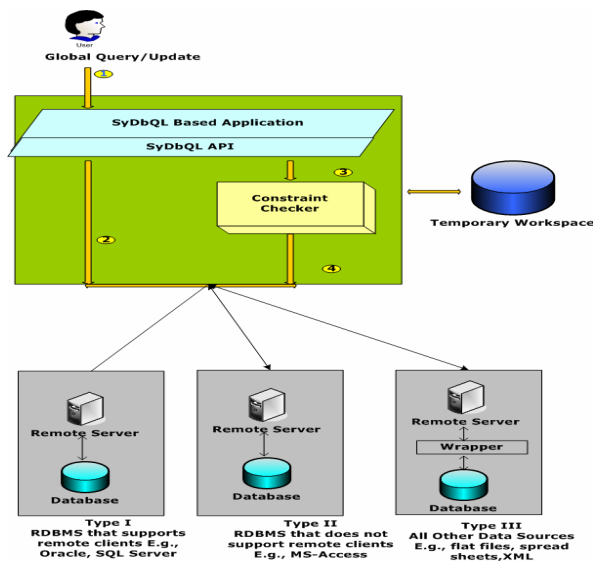


Figure 1. System of Databases Architecture

3. GLOBAL QUERIES AND SyDbQL SYNTAX

Consider the personal database which several individuals keep in their personal computers/personal digital assistants (PDAs). Typical data kept in these databases are appointments, addresses of acquaintances, etc. Let us assume there are three individuals, John, Tony and Aaron who maintain such data in their PCs/PDAs. John's database may have the following schema:

```
schedule (date, startTime, endTime,
event,status)
```

```
addressBook (name, email, address, wphone,
hphone, cphone, fax)
```

Tony's database may have the following slightly different schema:

```
schedule (date, startTime, endTime,
event,status)
addressBook (name, email)
```

Aaron's database may have the following schema, similar to John's except email addresses are not kept:

```
schedule (date, startTime, endTime, event,
status)
addressBook (name, address, wphone, hphone,
cphone, fax)
```

These three databases are assumed to be located on different nodes of a network and are assumed to be autonomous. Let us assume that these three individuals work together and hence a need for their databases to collaborate exists.

3.1 Global Queries

Global queries allow users in one database to extract information from their local database as well as remote databases. Such queries will have references to remote database objects. As an example of a global query, consider Aaron's problem of locating email addresses of all individuals in his addressBook. To accomplish this task, Aaron may execute the following *global query* in his database:

```
SELECT t.email
FROM tony.addressBook t, addressBook a
WHERE t.name = a.name;
```

Notice that in the above *global query* which executes within Aaron's database, there are references to tables in Tony. We are assuming the name of Tony's database is *tony*. Aaron is joining his addressBook table with that of Tony's to obtain email addresses.

3.2 SyDbQL Syntax

SyDbQL extends SQL by allowing tables to be referenced by the databases they are located in. A database naming mechanism is introduced, and tables in SyDbQL queries are identified by the databases they belong to. This is accomplished by preceding each table name with the database name of the table in the form of *database.table*. Following is a list of standard SyDbQL statements and their syntax:

a) Creating tables:

```
CREATE TABLE [dbname.]<tablename>[,
[dbname.]<tablename>]* (
col-def, ..., col-def, table-constr, ..., tab-
constr);
```

where col-def is:

```
<column-name><data-type>[DEFAULT
<expr>][<column-constraints>]
```

and tab-constr is:

```
[CONSTRAINT <constraint_name>] [NOT] NULL |
CHECK (<condition>) | UNIQUE | PRIMARY KEY |
REFERENCES
[dbname.]<table_name>[(<column_name>)] [ ON
DELETE CASCADE]
```

This statement allows one or more tables with the same schema to be created in remote databases. Constraints on table(s) can also be specified after CONSTRAINT keyword such as primary key, foreign key (REFERENCES), and cascading delete constraints.

b) Deleting tables:

```
DROP TABLE [dbname.]<tablename> [,
dbname.<tablename>]* [CASCADE
CONSTRAINTS];
```

This statement allows one or more tables to be deleted from multiple database schemas. CASCADE CONSTRAINTS allows the user to delete referenced tables as well.

c) Inserting rows into tables:

```
INSERT INTO [dbname.]<tablename>[,
dbname.<tablename>]* [(column {,column})]
VALUES (expression, {,expression});
```

Same row is inserted to one or more distributed database tables using SyDbQL INSERT statement.

d) Selecting rows from table(s):

```
<sub-select>
{ UNION [ALL] <sub-select> }[ ORDER BY
result_column [ASC | DESC ]
{ , result_column [ASC | DESC ]}]
```

```
where <sub-select> is:
SELECT [DISTINCT] <expression>
{,<expression>}
FROM [dbname. ]<tablename>[<alias>]
{,dbname. ]<tablename>[<alias>]}
[WHERE <search_condition>]
[GROUP BY <column> {,<column>}]
[HAVING <condition>]
```

SyDbQL SELECT statement is similar to standard SQL SELECT statement, but it allows querying tables from distributed databases. UNION statement provides a mechanism to get the union of the results of two SELECT statements. ORDER clause allows sorting the results.

4. SyDbQL JAVA API AND IMPLEMENTATION

The API consists of four main classes: SyDDatabase, SyDSystem, SyDConnection, and SyDStatement, and a helper class: SyDQueryParser. A sample snapshot of the SyDbQL Java API is given in the Appendix. Due to space limitations, we are not able to present API here; interested readers can access it at:

<http://tinman.cs.gsu.edu/~cscpnmx/SyDbQL/index.html>

We have implemented SyDbQL as a Java package, named `sydql`. Users can develop Java-based SyDbQL applications rapidly using this package. SyDbQL package allows programmers to register, create remote databases, and query multiple databases with single SyDbQL queries.

The global query is sent to the query engine which parses and generates sub queries that are sent to remote databases. The sub queries are generated based on the tables that are referenced in the query. The workspace database is used to perform the final evaluation of the global query after the sub query results have

been fetched into the workspace database. If there are any changes to be made to the remote databases then the action is eventually performed. The steps are summarized below:

1. Parse the global query and identify the databases that need to be accessed in the system.
2. Split the entire query into individual sub queries and send them to the corresponding databases. We directly use the concepts developed in the literature [4,10], for decomposing a global query in to sub queries. A detailed description of the algorithm is given in [10].
3. Collect the data obtained from the execution of sub queries on remote databases onto a virtual local machine (workspace).
4. Modify the global query such that all the tables referred in the query are local to the virtual local machine (workspace).
5. Execute the modified (global) query to get the desired output.

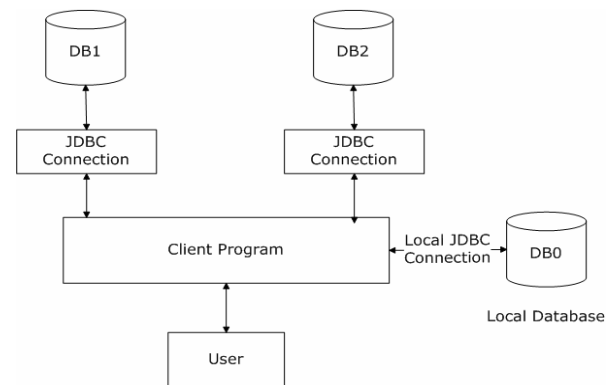


Figure 2. Local JDBC Connection Approach

We have used JDBC connection approach to send requests to remote databases and receive results back from them. In this approach, the client program executing in the workspace node makes direct JDBC connections to each of the remote databases that contain data relevant to the global query. The application then sends sub queries and retrieves results back into the workspace database where the final global query is executed. This approach is summarized in the Figure 2.

5. CALENDAR APPLICATION

The calendar of meetings application is an example of a typical collaborative application in which several individuals maintain their independent schedule information in their hand-held and other types of devices. The typical functionalities provided in such an application are: 1) set up meetings among individuals with certain conditions to be met such as a required quorum, 2) set up tentative meetings which could not be set up otherwise due to unavailability of certain individuals, and 3) remove oneself from a meeting or cancel an entire meeting resulting in automatic triggers being executed that may possibly convert tentative meetings into confirmed ones.

Such an application has been developed using the Java API presented in this paper. A snapshot of the sample Java code is shown below:

```
SyDSystem sydl = new SyDSystem();
sydl.addDatabase("john", databaseinfo);
sydl.addDatabase("tony", databaseinfo);
sydl.addDatabase("aaron", databaseinfo);
sydl.setMasterDB("john");
SyDConnection scl = sydl.getConnection();
SyDStatement ssl = scl.createSyDStatement();

// Prepare query to find idle times in each
// of three participants
// schedule on 03-March 2003.

// MasterDB is set to "john"; hence
//executes in john's local database
String query1 =
    "SELECT s.startTime, s.endTime " +
    "FROM   schedule s, tony.schedule ts,
aaron.schedule as " +
    "WHERE  s.date = ts.date and ts.date =
as.date and " +
    "as.date = to_date('03-MAR-
2003','DD-MON-YYYY') and " +
    "status='idle'";
ResultSet r1 = ssl.executeQuery(query1);

//process result set r1
//...
SyDConnection.close( );
```

The above code fragment shows how easy it is to gather information from a collection of autonomous databases and process it within the application. In this case, "idle" time slots in the schedules of three individuals are queried using a Global SyDbQL query. The complete calendar application has been implemented using Java servlets, oracle 9i database and SyDbQL API for global querying.

6. RELATED WORK AND CONCLUSIONS

Related Work: Considerable research results exists which extends the standard SQL for multidatabases or distributed databases [1,2,3,8,9,10]. Also, an abundant body of research has been directed towards the area of global query processing, global query decomposition and its optimizations [10,13]. An exhaustive survey of the state of the art in distributed query processing can be found in [7]. Our approach is directed towards building a user friendly Java API for SyDbQL. The Java API would ease the burden on the programmer by hiding all the complexities.

Most of the multidatabase systems such as MIND [4, 10], Pegasus [1], do not have a constraint checker module in built in their system. Instead they have a transaction control manager which is responsible for rolling back a transaction in case of a violation of a constraint. However, SyDb has a constraint checker module. The constraint checker [11,12] module in SyDb would check for global constraint violations before updating the database. In other words, the global update is executed only when there are no constraint violations. Hence our approach inherently optimizes on the global updates as there are no problems associated with rollbacks as such.

Conclusions: We have presented the design of the system of databases (SyDb) architecture for collaborative applications where querying across databases is frequent. A query language called SyDbQL is provided that basically extends SQL with database naming facilities. Main features of SyDbQL are: queries on all types of data formats and queries over a group of databases that form a system of databases.

A Java implementation of SyDbQL query processing package is also introduced. While implementing global queries/updates, the SyDbQL java API shaves off the burden from the application developer. We inherently achieve considerable optimization for global updates due to the constraint checker module proposed in the SyDb.

Future Work: Current implementation only queries Type I databases, i.e. JDBC and ODBC accessible data resources. The package has to be extended to work on the entire set of the heterogeneous databases, Type II and Type III databases (refer Section 2). The possible implementation for the Type-II databases (like MS-ACCESS) would be to run a remote server at the workstation that contains the Type-II database. The properties of the SydDatabase object of such database would include the remote server's name, the port on which it is running and the remote method to be called. The Type-III databases (like XML, spread sheets, etc..) can be included using the same method and with an additional wrapper that converts the data format to relational view and vice-versa. A topic for future work would be a wrapper system that will provide SQL access for data on Microsoft Excel sheets.

Also, the system works for a single global query/update. We have not addressed issues with global transactions. The possible implementation is to include a transaction monitor at each database that checks for the real time changes and maintains the ACID properties on each transaction. The transaction, which is an atomic operation, can be a set of SyDbQL statements i.e. either all the SyDbQL statements are executed or none of them has any action.

7. REFERENCES

- [1] R. Ahmed, P. De Smedt, W. Du, W. Kent, M. Ketabchi, A. Litwin, W. A., Rafii, and M. C. Shan. The Pegasus heterogeneous multidatabase system. IEEE Computer, 1991, pp. 19-27.
- [2] Y. Arens, C. A. Knoblock and W. Shen. Query Reformulation for Dynamic Information Integration. Journal of Intelligent Information Systems, 6(2/3), 1996, pp. 99-130.
- [3] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. Proceedings of IPSJ Conference, 1994
- [4] Dogac, A., Dengi, C., Kilic, E., Ozhan, G., Ozcan, F., Nural, S., Evrendilek, C., Halici, U., Arpinar, B., Koksall, P., Kesim, N., Mancuhan, S., METU Interoperable Database System, ACM SIGMOD Record, Vol.24, No.3, September 1995.
- [5] S. Grufman, F. Samson, S.M. Embury, P.M.D Gray and T. Risch : Distributing Semantic Constraints Between Heterogeneous Databases. Proceedings of ICDE 1997.

- [6] A. Gupta, and J. Widom. Local Verification of Global Integrity Constraints in Distributed Databases. Proceedings of the ACM SIGMOD, pages 49-58, Washington, D.C., May 1993.
- [7] Donald Kossmann. The state of the art in distributed query processing. .ACM Computing Surveys, vol. 32, no. 4, pp. 422-469, 2000
- [8] C. T. Kwok and D. S. Weld . Planning to gather information. Technical Report UW-CSE-96-01-04. Department of Computer Science, University of Washington, Seattle, WA, 1996.
- [9] Laks V. S. Lakshmanan and Fereidoon Sadri and Subbu N. Subramanian. SchemaSQL: An extension to SQL for multidatabase interoperability. ACM TODS, pp. 476-519, 2001
- [10] L.M. Mackinnon, D.H. Marwick, M.H. Williams. A Model for Query Decomposition and Answer Construction in Heterogeneous Distributed Database Systems. Journal of Intelligent Information Systems. 1998
- [11] P. Madiraju and R. Sunderraman. A Mobile Agent Approach for Global Database Constraint checking. Proceedings of the 2004 ACM Symposium on Applied Computing, Nicosia, Cyprus, pp. 679-683, 2004.
- [12] P. Madiraju and R. Sunderraman. An Efficient Constraint Planning Algorithm for Multidatabases. Accepted in 2005 ACS/IEEE International Conference on Computer Systems and Applications, Cairo, Egypt , Jan, 2005
- [13] F. Ozcan, S. Nural, P. Koksall, C. Evrendilek, A. Dogac: Dynamic Query Optimization in Multidatabases. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 1997.

8. APPENDIX

Generated Documentation (Untitled) - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Home Search Favorites Refresh

Address <http://tinman.cs.gsu.edu/~cscprnmx/SyDbQL/index.html> Go Links >>

Google Search Web Options

All Classes

- [SydConnection](#)
- [SydDatabase](#)
- [SydMetaData](#)
- [SydQueryParser](#)
- [SydStatement](#)
- [SydSystem](#)

Package [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Package java.sydq1

Class Summary

SydConnection	A connection (session) with a specific System of Database units (SydSystem).
SydDatabase	Single Database Unit of the System.
SydMetaData	An object that can be used to get information about the types and properties of the database units in the system Copyright: Copyright (c) 2001 Company. Yamacraw
SydQueryParser	This class parses the global query and identifies which tables to copy from which database units.
SydStatement	The object used for executing a static SydQL statement and obtaining the results produced by it This class handles the global query, sends the sub-queries to required databases, copies the tables, execute the queries Copyright: Copyright (c) 2001 Company. Yamacraw
SydSystem	This class maintains the actual system of databases.

Package [Class](#) [Tree](#) [Deprecated](#) [Index](#) [Help](#)

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Start C:\Do... acms... SyDb... Micro... ACM... EditP... acms... Adob... SyDb... Gene... 7:30 AM