A DATABASE INTEGRATION SYSTEM BASED ON GLOBAL VIEW GENERATION

Uchang Park
Duksung Women's University, Dobonggu Ssangmoondong 419, Seoul, Korea

Ramon Lawrence
University of British Columbia Okanagan, 3333 University Way, Kelowna, B.C., Canada

Keywords: Database, integration, view, heterogeneity.

Abstract: Database integration is a common and growing challenge with the proliferation of database systems, data warehouses, data marts, and other OLAP systems in organizations. Although there are many methods of sharing data between databases, true interoperability of database systems requires capturing, comparing, and merging the semantics of each system. In this work, we present a database integration system that improves on the database federation architecture by allowing domain administrators to simply and efficiently capture database semantics. The semantic information is combined using a tool for producing a global view. Building the global view is the bottleneck in integration because there are few tools that support its construction, and these tools often require sophisticated knowledge and experience to operate properly. The technique and tool presented is simple and powerful enough to be used by all database administrators, yet expressive enough to support the majority of integration queries.

1 INTRODUCTION

Research on how to integrate heterogeneous and autonomous database systems has been performed since the 1980's (Data Integration Projects, 2005). However, the early approaches, including federated databases (Sheth and Larson, 1990) and distributed databases did not solve the entire problem. In most cases, commercial databases like Oracle, SQL Server, and DB2 support proprietary integration of databases from the same vendor (Chawathe et al., 1994), (Collet et al., 1991), (IBM, 2005), (Josifovski et al., 2002), (MICROSOFT, 2005).

Mediator and wrapper architectures preserve database autonomy and support distributed queries. However, to be effective, a global view of the data sources must be created. Global view construction involves schema matching and merging techniques (Rahm and Bernstein, 2004). Although many prototype systems exist, the tools are not production-ready or easy to use by database administrators and designers. Further, there is no standard infrastructure, protocols, and implementation of the mediator and wrapper components. In this paper, we describe a system for helping database administrators to integrate databases and aids in the construction of a global view. Using the global view, we exploit the UnityJDBC integration system (Mason and Lawrence, 2005), (UNITYJDBC, 2006) for execution of the queries on the data sources. The UnityJDBC driver can integrate any number of JDBC-accessible data sources including SQL Server, MySQL, and Oracle. Our work uses techniques similar to those used in schema matching systems such as Clio (Haas et al., 2005) and COMA++ (Aumueller et al., 2005) and is related to the area of schema merging (Dragut and Lawrence, 2004).

The contributions of this work are:

- A global view construction technique that requires only knowledge of SQL.
- A query interface that allows users to write queries on the global view without understanding the underlying databases.
- A query execution system that automatically combines data from various sources according to the specified global view.

Park U. and Lawrence R. (2007).
A DATABASE INTEGRATION SYSTEM BASED ON GLOBAL VIEW GENERATION.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - DISI*, pages 453-456
DOI: 10.5220/0002351404530456

We overview the system architecture in Section 2. In Section 3, we discuss the basic constructs for building a global view. Section 4 contains related work, and the paper closes with future work and conclusions.

2 SYSTEM ARCHITECTURE

The overall system architecture is in Figure 1.

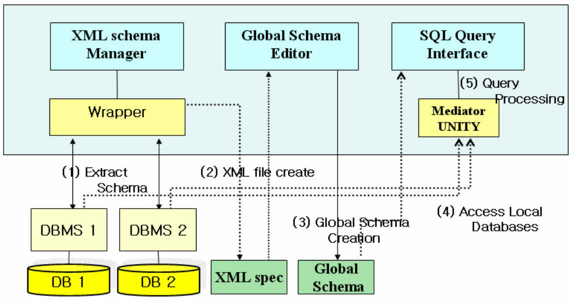


Figure 1: System Architecture.

The architecture provides a complete system for global view construction, data source querying and result integration.

Before the global view is constructed, the system collects each local database schema information and converts it to an XML file (Steps 1 and 2). The global schema editor is used to build the global view. Once constructed, the global view and mapping information is stored in a XML schema mapping document (Step 3). The mapping between local and global schemas is at the entity-level. That is, each table and attribute in a local view is mapped to a table or attribute in the global view. During query processing, the user specifies queries on the global view using a GUI. Global queries are converted into federation (source) queries. These source queries are executed by the federation engine (UnityJDBC) and then combined into a single result using the global view (Steps 4 and 5).

3 ARCHITECTURE DETAILS

3.1 Global View Construction

Our system connects to each local database and extracts the schema information into XML files. As an example, Figure 2 reports two simple Employee database schemas.

Our global view editor is shown in Figure 3. A global view is constructed by mapping the tables from the local databases to the global view. Once a set of global view relations are constructed using this bottom-up approach, the administrator uses the editor to construct local mappings. This mapping task is performed similar to schema matching approaches (Halevy, 2001), (Rahm and Bernstein, 2004) except that it allows more complex matching, including matching of relations as well as attributes.

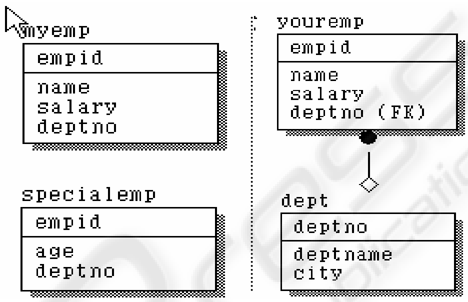


Figure 2: Site1 and Site2 Database Schema.

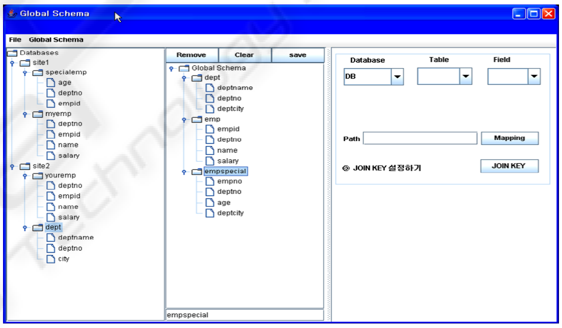


Figure 3: Global View Editor.

There are three types of attribute matchings:

- **Direct match** – When attribute B of the local schema matches exactly to attribute A of the global view.
- **No match** – Attribute B of a local schema has no matching in the global view.
- **Functional match** – Attribute A of the global view is matched by a functional expression (such as string concatenation) of one or more attributes in a single local schema.

For each relation in the local and global views, the primary keys of the relations are used to determine how to match relations. There are three ways to match relations between the local and global view:

- **Union** – The tuples of relation R in a local view are combined into relation S in the global view using UNION. There can be multiple relations that are unioned together to produce a global view relation instance.

- **Single** – Single is a special case of union where a global relation maps to only one local relation.
- **Join** – The tuples of a global relation S are produced by a join condition connecting relations R and U which may be in different sources.

As an example, Table 1 shows the mappings between a constructed global view and the schemas of the site 1 and site 2.

Table 1: Global to Local Mapping.

	Site 1	Site 2	Mapping
emp	**db1.myemp**	**db2.youremp**	**UNION**
empid	empid	empid	PK, Direct
name	name	name	Direct
salary	salary	salary	Direct
deptno	deptno	deptno	Direct
dept		**db2.dept**	**SINGLE**
deptno		deptno	PK, Direct
deptname		deptname	Direct
city		city	Direct
empspecial	**db1.specialemp**	**db2.dept**	**JOIN**
empno	empid		PK, Direct
deptno	deptno	deptno	Direct
age	age		Direct
deptcity		city	Direct

By constructing a global view, users are relieved of the burden of building distributed queries themselves which is typical in a database federation. The global schema is in Figure 4.

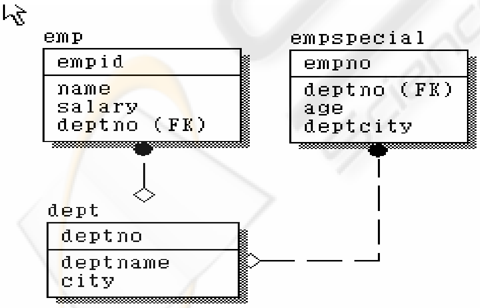


Figure 4: Global Schema for Employee Database.

3.2 Global View Querying

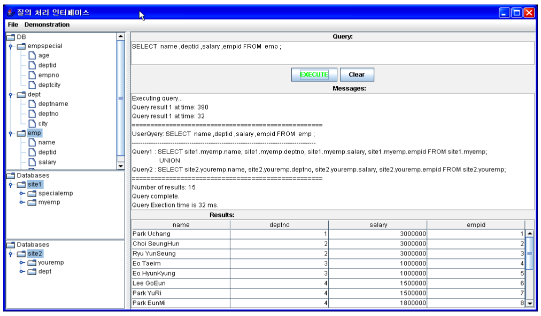
Queries on the integrated schema are built using a GUI interface (see Figure 5). Users can view the global schema while keying in queries over the screen. SQL queries are processed by a query processor that uses the XML mapping file and the execution engine. Data manipulation operations are not allowed on the GUI interface.

Depending on the relationship between a global schema table and a local schema table, the query handling rules are grouped into the following categories:

- **Type 1**: Query for single source table that comes from single local site.
- **Type 2**: Query for union table that comes from union of 2 or more local sites tables.
- **Type 3**: Query for join table that comes from join of 2 local sites tables.
- **Type 4**: Join query when joining 2 tables in the global view.

The rewriting rules for each type are as follows:

- **Type 1:** Map global relation and field names to local relation and field names.
- **Type 2:** Create a local source query for each source containing a table to be combined using union. Union each source query result. Union is applied only on the key attributes if the source tables have different numbers and types of fields.
- **Type 3:** Create a federated query that joins relations in the two databases on key attributes specified in the mapping.
- **Type 4:** A join in the global view may map to a UNION (Type 2) and a federated JOIN (Type 3) depending on the mapping for the table(s) in the global view. Multiple global joins may result in many UNIONs and federated joins in the federated query.



```
SELECT  name ,deptid ,salary ,empid FROM  emp ;
```

```
Executing query...
Query result 1 at time: 390
Query result 1 at time: 32
=====================================================
UserQyery: SELECT  name ,deptid ,salary ,empid FROM  emp ;
-----------------------------------------------------------------------------
Query1 : SELECT site1.myemp.name, site1.myemp.deptno, site1.myemp.salary, site1.myemp.empid FROM site1.myemp;
           UNION
Query2 : SELECT site2.youremp.name, site2.youremp.deptno, site2.youremp.salary, site2.youremp.empid FROM site2.youremp;
=====================================================
Number of results: 15
Query complete.
Query Exection time is 32 ms.
```

name	deptno	salary	empid
Park Uchang	1	3000000	1
Choi SeungHun	2	3000000	2
Ryu YunSeung	2	3000000	3
Eo Taeim	3	1000000	4
Eo HyunKyung	3	1000000	5
Lee GoEun	4	1500000	6
Park YuRi	4	1500000	7
Park EunMi	4	1800000	8

Figure 5: GUI Query Interface.

Once a query has been re-written from a global view query to a federated query, it is given to the federated query engine for execution.

The following are examples for each type based on our running example.

Type 1: Single Table

Global Query:

```
SELECT deptname
FROM dept
```

Federated Query:

```
SELECT db2.dept.deptname
FROM db2.dept
```

Type 2: Union

Global Query:

```
SELECT name
FROM emp
```

Federated Query:

```
SELECT db1.myemp.name
FROM db1.myemp
UNION
SELECT db2.youremp.name
FROM db2.youremp
```

Type 3: Federated Join

Global Query:

```
SELECT empspecial.empno,
        empspecial.deptcity
FROM empspecial
```

Federated Query:

```
SELECT db1.specialemp.empid,
       db2.dept.city
FROM db1.specialemp, db2.dept
WHERE db1.specialemp.deptno =
       db2.dept.deptno
```

Type 4: Global Join

Global Query:

```
SELECT emp.name, dept.deptname
FROM emp, dept
WHERE emp.deptno = dept.deptno
```

Federated Query:

```
SELECT db1.myemp.name,
        db2.dept.deptname
FROM db1.myemp, db2.dept
WHERE db1.myemp.deptno =
        db2.dept.deptno
UNION
SELECT db2.youremp.name,
        db2.dept.deptname
FROM db2.youremp, db2.dept
WHERE db2.youremp.deptno =
       db2.dept.deptno
```

4 CONCLUSIONS

Database integration offers benefits in three main areas: simplified system administration and maintenance, rapid development of integrated applications, and the ability for end-users to access all information in a domain. In this paper, we presented a database integration system that layers a global view on top of the federation architecture. This global view is simple to construct and maintain and allows federated queries to be automatically built by querying the global view. Thus, the approach captures the benefits of database federation, while avoiding its major shortcoming, the challenge of building federated queries to integrate data. Further, the approach, unlike commercial implementations, is not bound to a particular database management system. Overall, this makes the benefits of database integration easier and more cost-effective to realize in all organizations.

REFERENCES

Aumueller, D., Do, H.H.D., Massmann, S. and Rahm, E.R., 2005, Schema and Ontology Matching with COMA++. *In SIGMOD.*

Batini, C., Lenzerini, M., Navathe, S., 1986, A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys 18, 323.364*

Chawathe, S., et al., 1994, The TSIMMIS Project: Integration of Heterogeneous Information Sources, *Proceedings of the 10th Meeting of the Information Processing of Japan, pp. 7-18.*

Collet, C., Huhns, M., Shen, 1991, W.M.: Resource Integration Using a Large Knowledge Base in Carnot. *IEEE Computer 24, 55-62.*

Data Integration Projects World-Wide, 2005, http://www.ifi.unizh.ch/staff/pziegler/IntegrationProjects.html.

Dragut, E. and Lawrence, R., 2004, Composing Mappings between Schemas using a Reference Ontology. *In ODBASE, 783-800.*

Haas, L. M., Hernández, M.A., Ho, H., Popa, L. and Roth, M., 2005, Clio Grows Up: From Research Prototype to Industrial Tool. *In SIGMOD, 805-810.*

Halevy, A., 2001, Answering Queries Using Views: A survey. *VLDB Journal, 10 (4). 270-294.*

IBM, 2005, http://www.ibm.com, A Simple Introduction to Using DB2 Information Integrator with Oracle 9i.

Josifovski, V., Schwarz, P.M., Haas, L.M.H. and Lin, E.T., 2002, Garlic: A New Flavor of Federated Query Processing for DB2. *In SIGMOD, 524-532.*

Litwin, W., Mark, L. and Roussopoulos, M., 1990, Interoperability of Multiple Autonomous Databases. *ACM Computing Surveys, 22 (3). 267-293.*

Mason, T. and Lawrence, R., 2005, Dynamic Database Integration in a JDBC Driver. *In 7th International Conference on Enterprise Information Systems, 326-333.*

MICROSOFT, 2005, http://www.microsoft.com/korea/TechNet/biztalk/biztalka.asp

Rahm, R. and Bernstein, P., 2004, A Survey of Approaches to Automatic Schema Matching, *VLDB Journal, 10 (4). 334-350.*

Sheth, A., Larson, J., 1990, Federated Database Systems for Managing Distributed, Heterogenous and Autonomous Databases. *ACM Computing Surveys 22, 183-236.*

UNITYJDBC, 2006, http://www.unityjdbc.com/, UnityJDBC - Integrate SQL Data Sources using a Single Query.