5-1997

Source-aware multidatabase query processing

Ee Peng LIM
Singapore Management University, eplim@smu.edu.sg

Yinyan Cao

Roger Hsiang-Li CHIANG
University of Cincinnati

Source-aware Multidatabase Query Processing

Ee-Peng Lim, Yinyan Cao
School of Applied Science
Nanyang Technological University
Nanyang Avenue. Singapore 639798
E-mail: aseplim@sentosa.sas.ntu.sg

Roger H.L. Chiang
IMARC, Nanyang Business School
Nanyang Technological University
Nanyang Avenue. Singapore 639798
E-mail: ahlchiang@ntu.edu.sg

Abstract

We introduce a multidatabase model to represent the information that derives from different local databases. This model, known as Tuple-Source (TS) relational model, accommodates tuples from different local databases by attaching them with their source information in the global relations which are also known as TS-relations. In other words, a source attribute is implicit in every TS-relation. To manipulate the global relations, we have developed the TS-SQL query language and implemented a distributed query processor to process such queries. In this paper, we report our distributed query processing architecture and algorithms. Our architecture consists of a query mediator and a number of query agents, one for each local database. By encapsulating the heterogeneous query interfaces to different local databases within the query agents, we allow a generic query processing strategy to be adopted by the query mediator. Our approach further ensures that local autonomy is preserved.

1 Introduction

As databases become widely used in organizations, there is an increasing need to provide new global applications that access information from different databases. To facilitate the development of these new database applications, different kinds of multidatabase systems (MDBSs) have been proposed and implemented.

In our research, we propose to group MDBSs into three main categories:

- **Type (1):** These MDBSs choose not to handle any semantic heterogeneity, e.g.MSQL [LAZN89, WM90, LSS96]. In other words, they do not provide global integrated schemas over the pre-existing databases.
- **Type (2):** These MDBSs may support global integrated schemas but not integrated instances. In these MDBSs, the pre-existing database instances representing the same real-world objects are not entirely integrated together[AKWS95, LPL96].
- **Type (3):** These are MDBSs that integrated both the pre-existing database schemas and instances[CGH+94].

Since different MDBSs demonstrate different integration features, it is important for the users to select the appropriate MDBSs based on their global application requirement.

In this paper, we will describe a multidatabase model known as **Tuple Source (TS) Relational Model** that has been designed for type 2 MDBSs. As global schemas are supported, our multidatabase model hides the conflicts between local database schemas from the users. In addition, MDBS users do not need to remember the diverse structures in different local schemas. To achieve flexibility and extensibility, TS relational model requires every global relations to have an extra **source** attribute. By providing source attributes to all the global relations, we can identify the local database a global instance comes from. We can also relate local database semantics with the global instance so that further integration of global instances can be performed. An extended SQL language known as TS-SQL has been designed to query such global relations making use of the source information. The source information determines where the global tuples come from. It also allows global applications to access designated local databases through the global schemas. Our proposed TS relational model and TS-SQL query language can also be useful when a full integration of local database instances is too costly to perform. In this paper, we also describe the distributed query strategy for TS-SQL queries.

1.1 Related Research

In the following, we survey a number of type 2 multidatabase systems.

In [AKWS95], a multidatabase is defined to be a set of **flexible relations** in which local instances that represent the same real-world entities are stored together as groups of tuples. Hence, some implicit grouping of tuples in a flexible relation is required. Flexible relations also capture the source, consistency and selection information of their tuples. A corresponding set of flexible relational operations has been developed to manipulate the flexible relations. Nevertheless, flexible relational model is not a natural extension of the relational model. Furthermore, the join between flexible relations has not be defined.

A universal relational approach to model and query multidatabases is proposed in [ZSC95]. In this approach, a multidatabase is a universal relation instead of a set of relations. Queries on the universal relation are translated into multiple local queries against the local relations. The final query results are formed by unioning the local query results. Source information are attached to tuples in the final query results to indicate where the tuples come from. However, the source attribute is not included in the universal relation as well its query specification. Joins and other operations that involve multiple component databases are not allowed in this model.

1.2 Outline of Paper

The rest of the paper is structured as follows. We first present the unique features of the new TS-SQL language in Section 2. The referenced distributed query processing architecture is given in Section 3. We describe our distributed query processing steps in Section 3.2. Our proposed query decomposition strategy is discussed in detail in Section 4. Section 5 concludes the paper.

2 TS-Relational Model and TS-SQL Query Language

2.1 TS-Relations

Tuple-source (TS) relational model is specially designed for multidatabases that require global integrated schema but not integrated instances. For each global relation (TS-relation) in a multidatabase, a source value is attached to every tuple in the relation. In other words, every TS-relation comes with a mandatory source attribute. To manipulate TS-relations with extra source attributes, we have defined a set of source-aware TS-relational operations and the declarative query language, TS-SQL, which is an extension of the SQL (Please refer to [LC97] for a detailed discussion on TS-Relational Model). In the following, we use an integration scenario to illustrate the features of TS-SQL.

Multidatabase Example:

Let DB_A and DB_B be two local databases consisting of employee and department relations as shown below:

DATABASE DB_A

RELATION Emp_A

ename	dept	position	salary	qual
john	marketing	trainee	$1,000	Dipl.
mark	planning	manager	$3,000	B.Bus.
kim	marketing	secretary	$1,500	NULL
chen	marketing	engineer	$2,500	M.Eng.
daniel	library	engineer	$2,400	B.Eng.

RELATION $Dept_A$

dname	manager	floor	budget
marketing	chen	3	$2M
planning	mark	2	$4M
library	daniel	1	$1M

DATABASE DB_B

RELATION Emp_B

ename	dept	position	salary
john	research	trainee	$1,200
kim	marketing	secretary	$1,500
chen	marketing	leader	$2,600
stacy	marketing	sales rep	$3,500
sugimoto	research	fellow	$10,000
kain	research	engineer	$5,000

RELATION $Dept_B$

dname	manager	floor	budget
marketing	chan	3	$2.1M
research	sugimoto	6	$1M

To simplify our discussions, the above two local databases are assumed to have schemas compatible with the global schema of the multidatabase. By compatibility, we mean that real-world information are represented using the same schema constructs in the local database schemas and global schema. For example, the employee information in DB_A, DB_B and the multidatabase (its schema will be shown later) are represented in relations. Nevertheless, a local schema does not need to have a full set of relations found in the global schema. The local schemas can have missing attributes too. If the local database schemas are not compatible with the global schema, it is necessary to perform some database restructuring on the local databases when they are exported to the multidatabase system. Usually, such schema level incompatibility can be handled by the query agents or wrappers to be described in Section 3.

By combining the above relations, we obtain the following global relations for the multidatabase defined over DB_A and DB_B. Each global relation, in the form of TS-relations, consists of a source attribute with domain $\{DB_A, DB_B\}$. If an attribute in the global relation cannot be found in a corresponding local relation (e.g. qual), NULL values are assigned to the missing attribute for all tuples from that local relation. Note that the multidatabase is virtual, implying that any queries on the global relations have to be decomposed and disseminated to the relevant component databases and the results from them have to be combined to construct the global query results. In Section 2.2, we will describe how the semantics of the source attribute can be used in querying the multidatabase.

MULTIDATABASE GLOBAL RELATIONS

RELATION *Emp*

ename	dept	position	salary	qual	*source*
john	marketing	trainee	$1,000	Dipl.	DB_A
john	research	trainee	$1,200	NULL	DB_B
mark	planning	manager	$3,000	B.Bus.	DB_A
kim	marketing	secretary	$1,500	NULL	DB_A
kim	marketing	secretary	$1,500	NULL	DB_B
chen	marketing	engineer	$2,500	M.Eng.	DB_A
chen	marketing	leader	$2,600	NULL	DB_B
daniel	library	engineer	$2,400	B.Eng.	DB_A
stacy	marketing	sales rep	$3,500	NULL	DB_B
sugimoto	research	fellow	$10,000	NULL	DB_B
kain	research	engineer	$5,000	NULL	DB_B

RELATION *Dept*

dname	manager	floor	budget	*source*
marketing	chen	3	$2M	DB_A
marketing	chan	3	$2.1M	DB_B
planning	mark	2	$4M	DB_A
library	daniel	1	$1M	DB_A
research	sugimoto	6	$1M	DB_B

2.2 TS-SQL Query Language

In this section, we introduce TS-SQL which is used to specify declarative queries on TS-relations. TS-SQL is developed based on an extended relational model defined in [LC97]. The main features offered by TS-SQL include:

- TS-SQL satisfies the *closure property*. Given TS-relations, TS-SQL queries produce TS-relations as results.
- Unlike the traditional SQL, TS-SQL allows source options ($SAME\_DB$ and ANY_DB) to be specified on the SELECT and WHERE clauses, as well as on the union and intersect operations[1]. These options dictate how the tuples in the TS-relations are processed based on their source values. Source option specified on a SELECT clause determines if tuples from different local databases can be combined during projection. Source option specified on a WHERE clause determines if tuples from different local databases can be combined during join.
- Queries to specific local databases can be formulated by defining source predicates to operand TS-relations. A source predicate is represented by $< relation\_name > .sourcein < set\_of\_local\_DB\_IDs >$.
- Aggregation and groupby operations are customized to handle summarization of tuples based on their source values.

Despite TS-SQL resembles the traditional SQL closely, it can be shown that some of the TS-SQL queries involving the ANY_DB option cannot be performed both directly and indirectly by the normal SQL. Even when some of the TS-SQL queries can be computed by SQL expressions, we believe that TS-SQL will greatly reduce the effort of query formulation in the type 2 multidatabases.

Simple TS-SQL Queries:

Simple TS-SQL queries are SELECT-PROJECT-JOIN queries. In the following, we show a number of simple TS-SQL queries and explain their semantics.

Example:(Q1) Retrieve the name, salary and qualification of employees with salary less than $3,000 with reference to the local databases.

```
SELECT E1.ename, E1.salary, E1.qual [SAME_DB] FROM Emp E1
WHERE E1.salary < 3000
```

Q1's Result

E1.ename	E1.salary	E1.qual	*source*
john	$1,000	Dipl.	DB_A
john	$1,200	NULL	DB_B
kim	$1,500	NULL	DB_A
kim	$1,500	NULL	DB_B
chen	$2,500	M.Eng.	DB_A
chen	$2,600	NULL	DB_B
daniel	$2,400	B.Eng.	DB_A

With the source option $SAME\_DB$ assigned to the SELECT clause, Q1 requires the projection of Emp table to include the source attribute. Hence, tuples with identical projected attribute values but not source values remain to be separate in the query result, e.g. the information about kim.

[1] Due to space constraint, we will not present the union and intersect operations of TS-SQL here. Interested readers can refer to [LC97].

If the source information is not important during projection, the source option ANY_DB can be assigned to the SELECT clause as shown in Q2 below.

Example:(Q2) Retrieve the different positions held by employees regardless where the employee records come from.

```
SELECT E1.position [ANY_DB] FROM Emp E1
```

Q2's Result

E1.position	*source*
trainee	$*$
manager	DB_A
secretary	$*$
engineer	$*$
leader	DB_B
sales rep	DB_B
fellow	DB_B

As shown in Q2's result, positions that can be found in both local databases have $*$ as their source values. The $*$ value indicates that a tuple has been obtained by merging tuples from different local databases. In this case, we lose the source information of such tuples.

When two or more relations are given in the WHERE clause of TS-SQL, a source option can be assigned to the WHERE clause to indicate how the relations are to be joined together with respect to their source attributes.

Example:(Q3) Retrieve the employees and their managers according to the local databases they come from.

```
SELECT E1.ename, D1.manager [SAME_DB] FROM Emp E1, Dept D1
WHERE E1.dept=D1.dname [SAME_DB]
```

Q3's Result

E1.ename	D1.manager	*source*
john	chen	DB_A
john	sugimoto	DB_B
mark	mark	DB_A
kim	chen	DB_A
kim	chan	DB_B
chen	chen	DB_A
chen	chan	DB_B
daniel	daniel	DB_A
stacy	chan	DB_B
sugimoto	sugimoto	DB_B
kain	sugimoto	DB_B

Example:(Q4) Retrieve the employees and their managers regardless the local databases they come from.

```
SELECT E1.ename, D1.manager [ANY_DB] FROM Emp E1, Dept D1
WHERE E1.dept = D1.dname [ANY_DB]
```

By disregarding the source information, Q4 allows us to establish any possible relationship between the employees and managers across the local databases.

The source attribute can further allow us to direct queries to retrieve tuples from specified particular local database(s) as shown in Q5.

Example:(Q5) Retrieve the employees and their managers from local database DB_A[2].

[2]To conserve space, we do not show the result here.

```
SELECT E1.ename, D1.manager [SAME_DB] FROM Emp E1, Dept D1
WHERE E1.dept = D1.dname and *.source in {'DB_A'} [SAME_DB]
```

The predicate (*.source in {'DB_A'}) abbreviates ((E1.source in {'DB_A'}) and (D1.sources in {'DB_A'})). In Q5, as all tuples to be joined come from the DB_A, the source options assigned to SELECT and WHERE clauses can be ignored.

Aggregate and Groupby Queries

TS-SQL also supports aggregate and groupby queries. As shown in the following query examples, by assigning different source options to the SELECT clauses, we can obtain summarized information with or without reference to the source attributes.

Example: (Q6) Calculate the average salary for all employees in the Emp relation regardless of where they come from.

```
SELECT avg(E1.salary) [ANY_DB] FROM Emp E1
```

Q6's Result

avg(E1.salary)	*source*
3109	*

Example: (Q7) Calculate, for each department from each local database, the number of employees earning more than $2,000.

```
SELECT count(*), E1.dept [SAME_DB] FROM Emp E1
WHERE E1.salary > 2,000 GROUPBY E1.dept
```

Q7's Result

count(*)	E1.dept	*source*
1	planning	DB_A
1	marketing	DB_A
2	marketing	DB_B
2	research	DB_B

3 Distributed Query Processing Architecture

3.1 Query Mediator and Query Agent

As shown in Figure 1, our distributed query processor consists of a **query mediator** and a number of **query agents**, one for each local database.

The query mediator is responsible for decomposing global queries given by multidatabase applications into multiple subqueries to be evaluated by the query agents. It also assembles the subquery results returned by the query agents and further processes the assembled results in order to compute the final query result. Query agents transform subqueries into local queries that can be directly processed by the local database systems. The local query results are properly formatted before they are forwarded to the query mediator.

By dividing the query processing tasks between quey mediator and query agents, we allow concurrent processing of subqueries on local databases located at different sites, thus reducing the query response time. This architectural design further enables the query mediator to focus on global query processing and optimization while the query agents handle the transformation of subqueries decomposed by query mediator into local queries. Note that the query decomposition performed by query mediator assumes all local database schemas are compatible with the global schema. It is the job of query agents to convert the subqueries into local queries on possibly

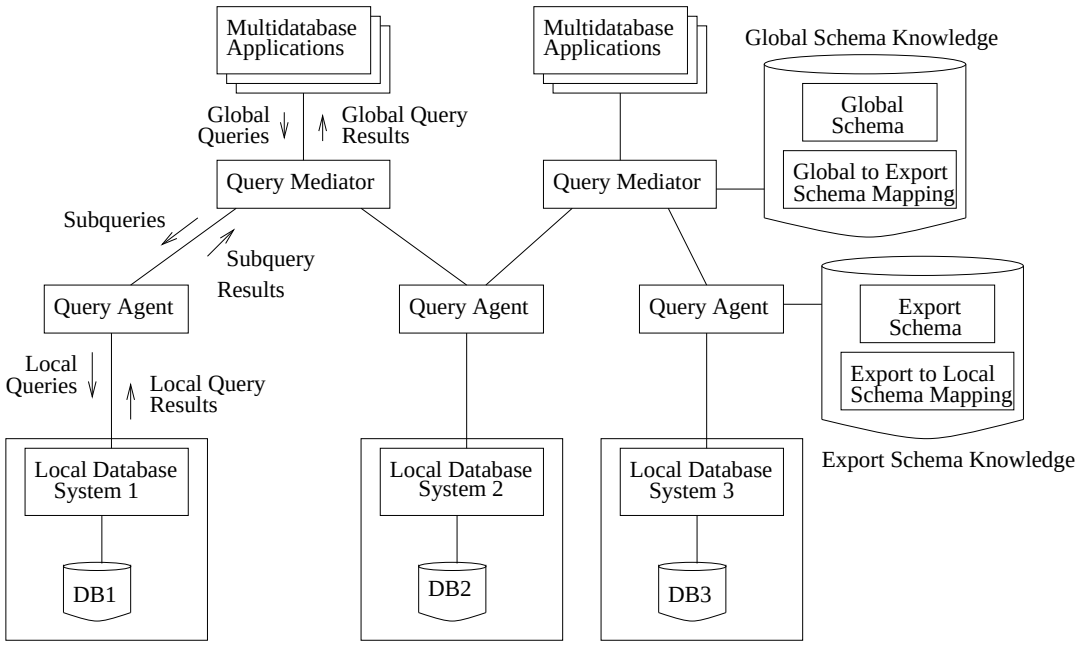

Figure 1: Multidatabase Query Processing Architecture

heterogeneous local schemas. The heterogeneous query interfaces of local database systems are also hidden from the query mediator by the query agents.

The query mediator and query agents require different types of schema knowledge for their query processing tasks. To decompose global queries into subqueries, the query mediator requires the global schema and global schema to export schema mapping knowledge. We define export schema to be the schema that is supported by a query agent. The export schema adopted by a query agent must be compatible with the global schema adopted by its query mediator. Similarly, the export schema and export to local schema mapping knowledge are information required by query agents to perform their query transformation.

3.2 Distributed Query Processing Steps

The overall distributed query processing steps designed for global TS-SQL queries is shown in Figure 2. In the following, we briefly describe the steps:

1. **Query Parsing:** Global TS-SQL queries are parsed to ensure that they are syntactically correct. Based on the parsed trees constructed, the queries are validated against the global schema to ensure that all relations and attributes in the queries exist and are properly used.

2. **Query Decomposition:** Given a global TS-SQL query, we decompose it into subqueries to be evaluated by the query agents. Here, the local databases involved in the global TS-SQL query will be determined. Some query optimization heuristics are introduced to reduce the processing overhead. Details of query decomposition will be given in Section 4.

3. **Local Query Evaluation:** Decomposed subqueries are disseminated to the appropriate query agents for execution. Query agents further translate the subqueries into local database queries and return the subquery results to the query mediator.

4. **Final Result Computation:** The query mediator assembles the subquery results and computes the final query result if there remain some query operations that could not be performed by the query agents. Examples of such query operations are attribute projection and aggregate functions in a SELECT clause with ANY_DB source option.

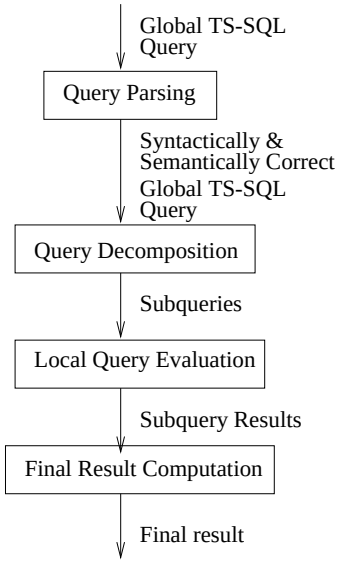

Figure 2: Distributed Query Processing Steps

4 Query Decomposition with Optimization

Unlike the traditional SQL queries, TS-SQL allows source options to be attached to their SELECT and WHERE clauses. Hence, our query decomposition strategy has to handle TS-SQL queries with different combination of source options. In decomposing global queries, our query decomposition strategy is designed to fulfil the following objectives:

- As far as possible, we would like the query agents to perform most query processing tasks in order to maximize the parallelism in local query evaluation.
- Heuristic query optimization has to be performed to reduce the subquery results as well as the local query results that have to be transferred from the local database sites to the query mediator. With small local query results shipped between sites, we can improve the query response time.
- Since not all TS-SQL operations can be performed by the local database systems, the decomposition process must consider the capabilities of query agents and also determine the portion(s) of global queries to be processed by the query mediator itself. At present, we have assumed that all query agents support the usual SELECT-PROJECT-JOIN SQL queries.

The source option on a SELECT clause determines if the source values of tuples have to be merged during the projection operation. On the other hand, the source option on a WHERE clause that involves more than one TS-relations determines how the join between the TS-relations is performed. In other words, if the source option SAME_DB is assigned to a WHERE clause, only tuples from the same local databases are allowed to be joined together. If the source option ANY_DB is assigned to a WHERE clause, tuples from any local database can be joined together.

By exploiting this join definition, we derive the decomposition strategies for the following two categories of TS-SQL queries:

- TS-SQL queries with SAME_DB assigned to their WHERE clauses, and
- TS-SQL queries with ANY_DB assigned to their WHERE clauses.

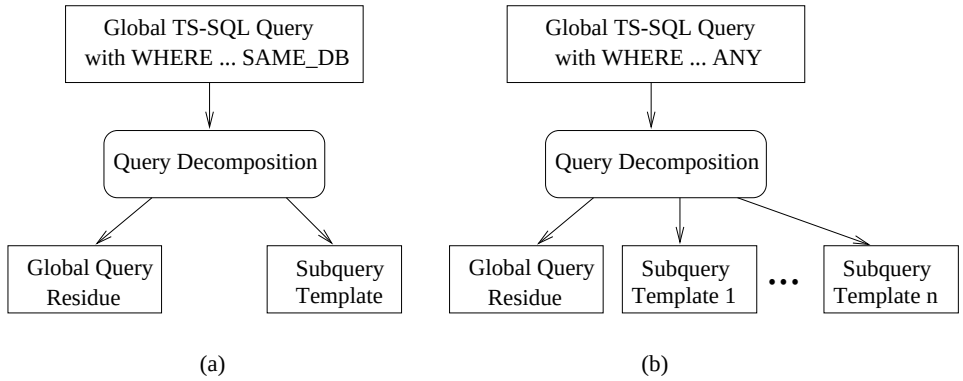

Figure 3: Distributed Query Processing Steps

4.1 Decomposition Strategy for TS-SQL Queries with "WHERE ... SAME_DB"

Given a global query in this category, we decompose it into a **subquery template** and a **global query residue** as shown in Figure 3(a). Here, the subquery template is a subquery generated based on the global schema and it has to be further translated into subqueries on the export schemas of local databases relevant to the global query. The global query residue represents the remaining global query operations that have to be handled by the query mediator.

Since SAME_DB is the source option of the WHERE clause, all selection and join predicates on the global TS-relation(s) can be performed by the query agents together with their local database systems. For example, in the following query Qa, we show that the join predicate ($E1.dept = D1.dname$) and selection predicate ($E1.salary > 3000$) have been propagated to the subquery template. Having performed a union of subquery results returned by the query agents, a final projection operation on the union result will be required as specified in the global query residue.

Global Query (Qa):
SELECT E1.ename, D1.manager [ANY_DB]
FROM Emp E1, Dept D1
WHERE E1.dept=D1.dname AND E1.salary > 3000 [SAME_DB]
Subquery Template:
SELECT E1.ename, D1.manager
FROM Emp E1, Dept D1
WHERE E1.dept=D1.dname AND E1.salary > 3000
Global Query Residue:
SELECT E1.ename, D1.manager [ANY_DB]
FROM < union of subquery results >

As shown in the above example, the subquery template involves global relation names and global attribute names. It has to be further translated into subqueries on the export schemas. In the following, we briefly describe the steps of deriving the subquery template and global query residue from a global query:

1. The SELECT clause of the subquery template is assigned the list of attributes that appears in the SELECT clause of the global query, including those appear in the aggregate functions[3].

[3]The decomposition can be further optimized by having the aggregate functions assigned to the subquery template when the SAME_DB option is attached to the SELECT clause of the global query.

2. The FROM clause of the subquery template is assigned the global relations that appear in the FROM clause of the global query.

3. Move the selection and join predicates in the WHERE clause of global query to the WHERE clause of the subquery template.

4. The global query residue inherits the SELECT clause of the original global query. Its FROM clause is defined by a union of subquery results. In other words, the only operations to be performed by the global query residue are projections, aggregations, and groupbys.

4.2 Decomposition Strategy for TS-SQL Queries with "WHERE ... ANY_DB"

For a global query in this category, our decomposition strategy generates a global query residue and multiple subquery templates, one for each global relation involved in the global query. This is shown in Figure 3(b). In other words, a global query with n relations in its FROM clause will be decomposed into n subquery templates. This is necessary because join predicates in the global query cannot be propagated to the subqueries.

For example, in the following query Qb, it can be shown that the join predicate ($E1.dept = D1.dname$) cannot be evaluated before the two global TS-relations Emp and $Dept$ are derived. Nevertheless, the selection predicate ($E1.salary > 3000$) can still be propagated to the subqueries for the local relations corresponding to Emp. Having performed unions of subquery results to construct the global relations Emp and $Dept$, a final join and projection on the global relations will be required as specified in the global query residue.

Global Query (Qb):
SELECT E1.ename, D1.manager [SAME_DB]
FROM Emp E1, Dept D1
WHERE E1.dept=D1.dname AND E1.salary > 3000 [ANY_DB]
Subquery Template 1 (for *Emp*):
SELECT E1.ename, E1.dept FROM Emp E1
WHERE E1.salary > 3000
Subquery Template 2 (for *Dept*):
SELECT D1.manager, D1.dname FROM Dept D1
Global Query Residue:
SELECT E1.ename, D1.manager [SAME_DB]
FROM < union of subquery results for *Emp* > E1,
< union of subquery results for *Dept* > D1
WHERE E1.dept = D1.dname [ANY_DB]

The steps of deriving the subquery templates and global query residue from a global query are:

1. For each global TS-relation (R) involved in the FROM clause, we generate its corresponding subquery template as follows:

   (a) The SELECT clause of the subquery template is assigned the list of R's attributes that appears in the SELECT clause of the global query, including those appear in the aggregate functions.

   (b) Selection and join predicates using R's attributes in the global query are propagated to the subquery template.

   (c) The FROM clause of the subquery template is assigned R.

2. The inter-global relation join predicates in the WHERE clause of global query, the projection and aggregation are retained in the WHERE clause of the global query residue.

4.3 Translation of Subquery Templates into Subqueries

In this section, we describe how the subquery templates decomposed from the global queries are translated into subqueries for the query agents. Although the query agents support subqueries on the export schemas which are compatible to the global schema, translating subquery templates into subqueries is still necessary for the following reasons:

- **Naming differences:** Schema elements in the export schema may be named differently from those in the global schema. For example, Emp in the subquery template has to be translated into subqueries involving Emp_A for local database A.
- **Missing export relations:** In our multidatabase query processor design, a global relation may not always has its corresponding export relation in every export schema. When a global relation in the FROM clause of a subquery template cannot be found in an export schema, no subquery will be generated for the corresponding query agent.
- **Missing attributes:** Some global attributes may not be found in the export schema. If any global attribute involved in a WHERE clause of a subquery template cannot be found in the export schema defined upon a local database, it is not required to translate the subquery template for the query agent of the local database.

5 Conclusions and Future Work

In this research, a distributed query processor for a multidatabase system based on TS-relational model has been developed. Using this multidatabase model, one can query the local databases via a global schema. The model also accommodates the local tuples in the global relation by attaching the source information to the tuples collected from multiple local databases. TS-SQL is the special query language designed for querying global TS-relations. This extended SQL language incorporates new features to manipulate the source attributes. We have also presented the processing strategy for source-aware TS-SQL queries over a collection of heterogeneous and autonomous local databases.

Our future research effort will focus on:

- *User friendly query interface:* We are now in the process of designing and implementing a user-friendly query interface for TS-SQL queries in the heterogeneous database environment. The query interface is expected to implemented on the web so that TS-SQL could be widely applied to public domain databases.
- *Queries to non-traditional databases:* Although TS-SQL has been original designed for querying distributed structured databases, we are looking forward to extending it to query widely available and possibly semi- or un-structured data on the internet.

References

[AKWS95] S. Agarwal, A.M. Keller, G. Wiederhold, and K. Saraswat. Flexible Relation: An Approach for Integrating Data from Multiple, Possibly Inconsistent Databases. In *Proceedings of the 11th International Conference on Data Engineering*, Taipei, Taiwan, 1995.

[ASD+91] R. Ahmed, P.D. Smedt, W. Du, W. Kent, M. Ketabchi, W.A. Litwin, A. Rafii, and M-C. Shan. The Pegasus Heterogeneous Multidatabase System. *IEEE Computer*, December 1991.

[CGH+94] D. Clements, M. Ganesh, S.-Y. Hwang, E.-P. Lim, K. Mediratta, J. Srivastava, J. Stenoein, and H. Yang. Myriad : Design and Implementation of a Federated Database Prototype. In *Proceedings of ACM SIGMOD Conference*, 1994.

[Hug94] D.J. Hughes. Mini SQL: A Lightweight Database Engine. Technical Report Version 1.0, December 1994.

[LAZN89] W. Litwin, A. Abdellatif, A. Zeroual, and B. Nicolas. MSQL: A Multidatabase Language. *Information Sciences*, 49:59–101, 1989.

[LC97] Ee-Peng Lim and Roger H.L. Chiang. TS Relational Model: A Data Model for Multi-databases with Integrated Global Schemas and Non-integrated Instances. Technical Report 1, School of Applied Science, Nanyang Technological University, January 1997.

[LPL96] L. Liu, C. Pu, and Y. Lee. An Adaptive Approach to Query Mediation Across Heterogeneous Databases. In *International Conference on Cooperative Information Systems*, Brussels, June 1996.

[LSS96] L.V.S. Lakshmanan, F. Sadri, and I.N. Subramanian. Schema SQL - A Language for Interoperability in Relational Multi-database Systems. In *Very Large Data Base Conference*, 1996.

[WM90] R. Wang and S. Madnick. A Polygen Model for Heterogeneous Database Systems: The Source Tagging Perspective. In *Very Large Data Base Conference*, pages 519–538, Brisbane,Australia, 1990.

[ZSC95] J.L. Zhao, A. Segev, and A. Chatterjee. A Universal Relation Approach to Federated Database Management. In *Proceedings of the 11th International Conference on Data Engineering*, Taipei, Taiwan, March 1995.