FTS-SQL: A Query Language for Fuzzy Multidatabases

Awadhesh Kumar Sharma[*], A. Goswami[†], D.K. Gupta[‡]

Abstract—**Once the relational data model is extended to support the fuzzy multidatabases, there is need to have a language to facilitate fuzzy queries formulations on integrated fuzzy relations under the extended relational data model. Formulation of such a query needs functions that do not exist in classical data manipulation languages such as SQL, which is designed to manipulate data in a single database. Classical SQL deals with single relation as an object of manipulation where as the current need is to allow sets of relations as the objects. Therefore, there is a need to extend the SQL to FTS-SQL that supports fuzzy query formulation on FTS relations [12] under the extended relational data model. To extend the SQL to FTS-SQL, a query syntax is proposed with additional clause HAVING α, β and WITH SOURCE-CONTEXT (optional). An option [sameDB] or [anyDB] is attached to SELECT and WHERE clauses that give a direction to the query processor for query evaluation. By specifying a suitable value to $\alpha, \beta \in [0, 1]$, a precision fuzzy query can be formulated. Default values for α and β are set to 1 which correspond to crisp case. Thus formulation of crisp queries on FTS relations are also allowed. Use of WITH clause joins the source context information to each of the resultant tuple. To process these fuzzy queries efficiently, a query processing architecture is proposed that has a query mediator and a number of query agents to handle the query processing tasks. A query decomposition and optimization strategy is proposed based on the use of option [sameDB]/[anyDB] attached with WHERE clause and two algorithms namely, FTS-SQL-Processing-1 & FTS-SQL-Processing-2 are devised to support them. The query syntax and the query processing algorithms are explained by means of illustrative examples.**

Keywords: fuzzy multidatabases, fuzzy query formulation, fuzzy query processing, fuzzy query optimization

1 Introduction

Many users usually have their data in several databases and frequently need to jointly manipulate data from different databases. For example: a traveller looking for cheapest route may need to query several airline, rail, and bus databases. The manager of a company may need to see the account balances that the company has at different bank branches. Therefore, he would like to query all the relevant bank databases and his own databases.

[*]MMMEC, Gorakhpur, UP, India, email:akscse@rediffmail.com
[†]IIT Kharagpur, WB, India, goswami@maths.iitkgp.ernet.in
[‡]IIT Kharagpur, WB, India, dkg@maths.iitkgp.ernet.in

Formulation of such a query needs functions that do not exist in classical data manipulation languages such as SQL, which is designed to manipulate data in a single database. Classical SQL deals with single relation as an object of manipulation where as the current need is to allow sets of relations as the objects. A system for the manipulation of data in autonomous databases is called a multidatabase system (MDBS) and the corresponding language is called a multidatabase manipulation language (MML) [9]. Databases may be manipulated together without global integration are called interoperable [5]. [6] present the multidatabase extension of SQL called MSQL that contains new functions designed for nonprocedural manipulation of data in different and nonintegrated SQL databases. A theoretical foundation for such languages has been proposed in [2] by presenting a multirelational algebra and calculus based on relational algebra and calculus. In [7] a semantic query language, SemQL, has been provided to enable users to issue queries to a large number of autonomous databases without prior knowledge of their schema. [10] presents an unique approach to query decomposition in a multidatabase environment which is based on performing transformations over an object algebra that can be used as the basis for a global query language.

Many real world problems involve imprecise and ambiguous information rather than crisp information. Recent trends in the database paradigm are to incorporate fuzzy sets to tackle imprecise and ambiguous information of real world problems. Query processing in multidatabases have been extensively studied, however, the same has rarely been addressed for fuzzy multidatabases. In this paper, an attempt is made to extend the SQL to formulate a global query on a fuzzy multidatabase under FTS relational model discussed in [12]. An architecture for distributed query processing with a strategy for query decomposition and optimization has also been provided.

1.1 Assumptions

The proposed system configuration basically consists of one global site and a number of local sites. One of the local sites can be the global site, in which case a data communication cost is saved for the site and the global site. Each local site maintains its own data management system that support FSQL [3, 4] and is independent of other sites. A user communicates only with the global site. A global query using FTS-SQL is entered at the global site, and results are received from the

global site. The global site maintains information about each local fuzzy database structure, such as fuzzy schema definitions and which kind of fuzzy relations are stored in which local fuzzy database. This allow the global site to efficiently schedule a global fuzzy query processing plan. Each local fuzzy database can accommodate any number of fuzzy relations and optimally process each local fuzzy query given by the global site.

Rest of the paper is organized as follows: Section 2 gives brief definitions of the concepts used in developing the algorithm in question. Section 3 introduces and discusses an algorithm to map the problem of Discovery of Fuzzy Inclusion Dependencies (FID_α) in Fuzzy Databases to the problem of Finding Cliques in Hypergraphs. The algorithm for the Discovery of FID_α is developed and discussed in section 4 and finally the paper is concluded in section 5.

2 The Query Syntax

Well defined semantics of FTS-relational operations may not be directly suitable for query formulation, hence FTS-SQL (the fuzzy version of TS-SQL [8]) is designed. Demonstration of syntax of a simple FTS-SQL query may be given as follows:

SELECT	$< target\ attributes >$[anyDB]/[sameDB]
WITH	Source Context{optional}
FROM	$< FTS - relation >$
WHERE	$< selection/join$
	$conditions >$ [anyDB]/[sameDB]
HAVING	$< value\ of\ \alpha, \beta >$

It may be noted that the keyword anyDB or sameDB are optional for SELECT clause and WHERE clause and at the same time a simple FTS-SQL query has an optional WITH clause and HAVING clause which makes it different from normal SQL. For both clause, default keyword used is anyDB. If WITH clause is specified in a FTS-SQL query, the query result will also include the source context allowing the user to interpret the query tuples using source context. The HAVING clause specifies the value of $\alpha, \beta \in [0, 1]$ where α is used for α-resemblance [11] of fuzzy attribute values and β is used as a threshold of the membership grade of fuzzy tuple to the fuzzy relation to satisfy the *selection* criteria. A FTS-SQL query can be written as:

SELECT	R.A, S.B [sameDB]
FROM	R,S
WHERE	R.X EQ S.X AND R.U EQ 'abc'
	AND S.V EQ HIGH [anyDB]
HAVING	0.8, 0.6

This query can be translated into an equivalent FTS-relational expression as shown below:

$$\overset{fs}{\pi}{}^{sameDB}_{R.A,S.B,.8,.6}\Big(\big(\overset{fs}{\sigma}{}_{R.U\ EQ\ 'abc'.8,.6}\ R\big)\ \overset{fs}{\bowtie}\ \overset{anyDB}{}_{R.X\ EQ\ S.X,.8,.6}\big(\overset{fs}{\sigma}{}_{S.V\ EQ\ High,.8,.6}S\big)\Big)$$

This translation is allowed because both $\overset{fs}{\bowtie}\ \overset{sameDB}{}_{\alpha,\beta}$ and are $\overset{fs}{\bowtie}\ \overset{anyDB}{}_{\alpha,\beta}$ commutative and associative, however, source predicate must be evaluated before any FTS join is carried out, since attributes of operand relations are merged during the join. A FTS-SQL query involving union, intersection or subtraction of two or more FTS-relation, can be written as given below:

SELECT	$< target\ attributes >$[anyDB]/[sameDB]
WITH	SOURCE CONTEXT{optional}
FROM	$< FTS - relation >$
HAVING	$< value\ of\ \alpha, \beta >$
WHERE	$< selection/join$
	$conditions >$[anyDB]/[sameDB]
Union	[anyDB]/[sameDB] or
Intersection	[anyDB]/[sameDB] or
Minus	[anyDB]/[sameDB]
SELECT	$< target\ attributes >$[anyDB]/[sameDB]
WITH	SOURCE CONTEXT{optional}
FROM	$< FTS - relation >$
WHERE	$< selection/join$
	$conditions >$[anyDB]/[sameDB]
HAVING	$< value\ of\ \alpha, \beta >$

In summary, the main features offered by FTS-SQL include:

- FTS-SQL satisfies the closure property. Given FTS-relations, queries produce FTS-relations as result.
- FTS-SQL allows source options [anyDB]/[sameDB] to be specified on the SELECT and WHERE clauses, as well as on the *union, intersection & minus* FTS operations. Source option on SELECT clause determines if tuples from different local databases can be combined during projection. Source option on WHERE clause determines if tuples from different local databases can be combined during join.
- Defining source predicates to operand FTS-relations, can make formulation of queries to a specific local database. A source predicate is represented by $<$ $relation\ name > .source$ in <set of local $DB_{id} >$
- Queries can be on both crisp and fuzzy attributes. By specifying different values of α, β in the HAVING clause, the precision in query formulation can be adjusted. $\alpha \in [0, 1]$ is used for α-resemblance of fuzzy values in fuzzy predicates, where as $\beta \in [0, 1]$ imposes a constraint while selecting a fuzzy tuple. Default value for both of them is 1 which correspond to crisp values. It can be shown that some of the FTS-SQL queries involving the anyDB option can not be performed both directly or indirectly by the normal SQL. Even when some

of the FTS-SQL queries can be computed by FSQL expressions, it is believed that FTS-SQL will greatly reduce the effort of query formulation on fuzzy relational multidatabase of type-2.

- Using the clause WITH SOURCE CONTEXT, tuples in the query results can be joined with its source related information available in *source relation* table 3.

Table 1. Set of export fuzzy relations from databases DB_1 & DB_2

$[DB_1]$: Emp_1

Name	Age	Hall	μ_r
Jaya	.5/old	MT	.50
Apu	.5/mid	JCB	.50

$[DB_1]$: $Dept_1$

Dname	HoD	Fund	μ_r
Chem	Jaya	.63/low	.63
Eco	Maya	.63/mod	.63

$[DB_2]$: Emp_2

Name	Age	μ_r
Jaya	.5/mid	.50
Maya	.5/mid	.50

$[DB_2]$: $Dept_2$

Dname	Staff	HoD	Fund	μ_r
Eco	10	Maya	.6/mod	.60
Chem	15	Jaya	.63/mod	.63

In the following examples, it is shown that a number of simple global fuzzy queries can be formulated using FTS-SQL over the FTS relations given in Table 2 and their semantics are explained . In every example it is assumed that $\alpha = .8$ and $\beta = .6$ to indicate the precision of the fuzzy query.

Example 1 *Q_1: Retrieve the Department name and the name of head of the department who has the departmental fund greater than .6/mod.*

SELECT *T.Dname, T.HoD, T.Fund[sameDB]*
FROM *Dept T*
WHERE *T.Fund EQ_α .66/mod*
HAVING *.8,.6*

T.Dname	*T.HoD*	*T.Fund*	μ_r	*source*
Chem	*Jaya*	*.63/low*	*.63*	DB_1
Eco	*Maya*	*.63/mod*	*.63*	DB_1
Eco	*Maya*	*.6/mod*	*.6*	DB_2
Chem	*Jaya*	*.63/mod*	*.6*	DB_2

With the source option [*sameDB*] assigned to SELECT clause, Q_1 requires the projection of *Dept* to include the source attribute. Hence the fuzzy tuples with the identical projected attribute values not source values remain to be separate in the

Table 2. FTS Relations: level-0 integration of DB_1, DB_2

Emp=merge(Emp_1, Emp_2)

Name	Age	Hall	μ_r	source
Jaya	.5/old	MT	.50	DB_1
Apu	.5/mid	JCB	.50	DB_1
Jaya	.5/old	Null	.50	DB_2
Maya	.5/mid	Null	.50	DB_2

$Dept$=merge($Dept_1, Dept_2$)

Dname	Staff	HoD	Fund	μ_r	source
Chem	Null	Jaya	.63/low	.63	DB_1
Eco	Null	Maya	.63/mod	.63	DB_1
Eco	10	Maya	.6/mod	.60	DB_2
Chem	15	Jaya	.63/mod	.63	DB_2

Table 3. Source relation to provide source (context) Semantics

Source Relation

source	Org-Unit	LOC	DBMS	DBA
DB_1	Personnel	IITD	ORACLE	Gupta
DB_2	R & D	IITK	DBASE3	Nanda

query result, e.g. information about *Eco* department. The two fuzzy values .6/mod and .63/mod are α-resemblant but the tuples related with Eco department are not merged rather remain to be separate in the query result. If the source option [*anyDB*]is not important during the projection, the source option can be assigned to the SELECT clause as shown in the next query example.

Example 2 *Q_2: Retrieve the Department name and the departmental fund regardless where the department record come from.*

SELECT *T.Dname, T.Fund[anyDB]*
FROM *Dept T*
HAVING *.8,.6*

T.Dname	*T.Fund*	μ_r	*source*
Chem	*.63/low*	*.63*	DB_1
Eco	*.63/mod*	*.63*	*
Chem	*.63/mod*	*.6*	DB_2

As shown in result of the query Q_2, fuzzy tuples that are α-resemblant are merged using fuzzy *union* and the source value of the merged tuple is indicated by '*' if the parent tuples come from different source.

3 Distributed Query Processing Architecture

3.1 Query Mediator and Query Agents

The proposed distributed query processor has a query mediator and for each local database there is one query agent. The

responsibilities of a query mediator are:

1. to take the global queries as input given by multi-database applications, and decompose it into multiple sub-queries to be evaluated by the query agents of the respective local databases. For this decomposition process it has to refer to Global Fuzzy Schema to Export Fuzzy Schema Mapping information. This unique information is supposed to be stored in the FMDBS.
2. to forward the decomposed queries to respective local query agents.
3. to assemble the sub-query results returned by query agents and further process the assembled results in order to compute the final query result.
4. to transform back the format of final query result into a format that is acceptable to multi-database applications. Here again it refers to Global fuzzy Schema to Export Fuzzy Schema Mapping information.

The responsibilities of a query agents are:

1. to transform sub-queries into local queries that can be directly processed by the local database systems. This transformation process refers to Export Schema and Export to Local Schema Mapping information. This information is supposed to be stored in respective local databases.
2. to transform back (using Export Schema and Export to Local Schema Mapping information) the local query results into a format that is acceptable to the query mediator and forward the formatted results to the query mediator.

Sub-queries are independent hence they may be processed in parallel at respective local databases. This reduces the query response time. Query agents hide heterogeneous query interfaces of local database systems from the query mediators.

Distributed query processing steps designed for global FTS-SQL queries can be described briefly as follows: Global FTS-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. Given a global FTS-SQL query, the query mediator decomposes it into sub-queries to be evaluated by the query agents. Here the local database involved in global FTS-SQL query will be determined. Some query optimization heuristics are introduced to reduce the processing overhead. Similar strategies have been adopted for optimizing queries for other multi-database systems [1]. Decomposed sub-queries disseminated to the appropriate query agents for execution. Query agents further translate the sub-queries to the local database queries and return the sub-query results to the query mediator. The query mediator assembles the sub-query results and computes the final query result if there exist some sub-query operations that could not be performed by the query agents.

4. Query Decomposition with Optimization

FTS-SQL allows source options to be attached to their SELECT and WHERE clauses. Hence a strategy should should be evolved to decompose a FTS-SQL query to handle different combination of source options. Therefore, it has been designed to meet the following objectives:

(a) Query agents are supposed to perform most of the query processing tasks in order to maximize the parallelism in local query evaluation.

(b) To reduce the sub-query results that have to be transferred from the local database sites to the query mediator and heuristic query optimization has to be performed. Query response time can be improved with small local query results shipped between sites.

(c) Since every FTS-SQL operations can't be performed by 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 it is assumed that all query agents support the usual select-project-join FTS-SQL queries.

WHERE clause with the source option sameDB allows the join of tuples from the same local database only and with the source option anyDB it allows the join of tuples from any local database. Based on this join definition, the decomposition strategies are derived for following two categories of FTS-SQL queries:

4.1. FTS-SQL queries with WHERE $<$ $\ldots$ $>$[sameDB]

As per this strategy, a global query is decomposed into a sub-query template and a global query residue. Sub-query template is the sub-query generated based on the global schema and it has to be further translated into sub-queries on the export schemas of local fuzzy 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 *sameDB* is the source option of the WHERE clause, all *selection* and *join* predicates on the global FTS relation(s) can be performed by the query agents together with their local database systems. Deriving the *Sub-Query Template* and the *Global Query Residue* from a *Global Query* may be given as follows:

1: The SELECT clause of Sub-Query Template is assigned the list of attributes that appear in the SELECT clause of Global Query, including those which appears in the *aggregate* functions.

2: The FROM clause of Sub-Query Template is assigned the global FTS-relations that appear in the FROM clause of Global query.

3: Move the selection and join predicates in the WHERE clause of Global Query to the WHERE clause of Sub-Query Template.

4: The Global Query Residue inherits the SELECT clause of the original Global Query. Its FROM clause is defined by the union of sub-query results. In other words, only operations to be performed by the Global Query Residue are Projections. The WITH clause is retained in the Global Query Residue and performed in the last phase of the query processing.

5: The values of α and β in the HAVING clause of the global query are assigned to the having clause of all the sub-queries.

Example 3 *Consider the global fuzzy database as given in Table 2 whose component local fuzzy databases are given in Table 1. In the following query Q_a, it is shown that the join predicate ($T_1.Name\ EQ_\alpha\ T_2.HoD$) and selection predicate ($T_2.Fund\ EQ_\alpha$ .66/mod) have been propagated to the sub-query template for decomposition of a global query that has been formulated using FTS-SQL. Having performed a union of sub query 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(Q_a):
SELECT $T_1.Name, T_2.Dname$ *[anyDB]*
FROM *Emp* T_1, *Dept* T_2
WHERE $T_1.Name\ EQ_\alpha\ T_2.HoD$ *AND*
$T_2.Fund\ EQ_\alpha$ .66/mod *[sameDB]*
HAVING *.8, .6*

Sub-Query Template:
SELECT $T_1.Name, T_2.Dname$
FROM *Emp* T_1, *Dept* T_2
WHERE $T_1.Name\ EQ_\alpha\ T_2.HoD$ *AND*
$T_2.Fund\ EQ_\alpha$ .66/mod
HAVING *.8, .6*

Global Query Residue:
SELECT $T_1.Name, T_2.Dname$ *[anyDB]*
FROM *<Union of sub-query results>*
HAVING *.8, .6*

4.2. FTS-SQL queries with WHERE $< \ldots >$[anyDB]

This strategy generates one Global Query Residue and multiple Sub-Query Templates, one for each global relation involved in the Global Query. In other words, a Global Query with n relations in its FROM clause will be decomposed into n Sub-Query Templates. This is necessary because join predicates in Global Query can't be propagated to the Sub-Queries. Given below is the sequential steps to derive *Sub-Query Templates* and *Global Query Residue* from a *Global Query*.

1: For each global FTS-relation R involved in the FROM clause its corresponding sub-queries are generated as follows:

(a) The SELECT clause of Sub-Query Template is assigned the list of R's attributes that appears in the SELECT clause or join predicates of the Global Query, including those which appears in the *aggregate*functions.

(b) Selection and join predicates using R's attributes in the Global Query are propagated to Sub-Query Template.

(c) The FROM clause of Sub-Query Template is assigned R.

2: For the inter-global relation *join* predicates in the WHERE clause of Global Query, the *projections* are retained in WHERE clause of Global Query Residue. The clause WITH SOURCE CONTEXT is also retained, however, processed at last.

3: The values of α and β in the HAVING clause of the global query are assigned to the having clause of all the sub-queries.

Example 4 *Consider here again a similar query as in Example 3 but with* anyDB *option attached to WHERE clause. Thus, a new global query Q_b has been formulated which is decomposed as shown below:*

Global Query(Q_b):
SELECT $T_1.Name, T_2.Dname$ *[sameDB]*
FROM *Emp* T_1, *Dept* T_2
WHERE $T_1.Name\ EQ_\alpha\ T_2.HoD$ *AND*
$T_2.Fund\ EQ_\alpha$ .66/mod *[anyDB]*
HAVING *.8, .6*

Sub-Query Template-1(for Emp):
SELECT $T_1.Name$
FROM *Emp* T_1
HAVING *.8,.6*

Sub-Query Template-2(for Dept):
SELECT $T_2.Dname, T_2.Fund, T_2.HoD$
FROM *Dept* T_2
WHERE $T_2.Fund\ EQ_\alpha$ .66/mod
HAVING *.8,.6*

Global Query Residue:
SELECT $T_1.Name, T_2.Dname$*[sameDB]*
FROM *<union of Sub-query results for Emp>* T_1,
<union of Sub-query results for Dept> T_2
WHERE $T_1.Name\ EQ_\alpha\ T_2.HoD$
HAVING *.8,.6*

The FTS-SQL query shown in example 4, it can be shown that the join predicate $T_1.NameEQ_\alpha T_2.HoD$ can't be evaluated before the two Global FTS-relations Dept and Emp are derived. Nevertheless, the selection predicate $T_2.FundEQ_\alpha.66/mod$ can still be propagated to the sub-queries for local relations corresponding to Dept. Having performed unions of sub-query results to construct the global relations $Dept$ and Emp, a final *join* and *projection* of the global relations will be required as specified in the Global Query Residue.

5 Conclusions and Future Work

In this paper a query language FTS-SQL is proposed to formulate a global fuzzy query on a fuzzy relational multidatabase of type-2 under FTS relational model. *FTS relational operations* operate on FTS relations to produce a resultant FTS relation. An architecture for distributed query processing with a strategy for query decomposition and optimization is also provided. Sub-queries obtained as a result of query decomposition are allowed to get processed in parallel at respective local fuzzy databases. This reduces the query processing time effectively.

The future work will explore an appropriate extension to the relational model for integrated fuzzy databases with level-1 instance integration using information on fuzzy inclusion dependencies among component fuzzy relational databases. A user-friendly query interface can be designed and developed for FTS-SQL queries in the heterogeneous database environment.

References

[1] Evrendilek, C., Dogac, A., Nural, S., Ozcan, F., "Query optimization in multidatabase systems", *Jouranal of Distrubuted and Parallel Databases*, 5(1) (1997), 77-114.

[2] John Grant, Witold Litwin, Nick Roussopoulos, Timos Sellis, "Query languages for relational multidatabases", *The VLDB Journal*, Volume 2, Issue 2 (1993), pp 153-172.

[3] Galindo J., Medina J.M., Cubero J.C., Garca M.T., "Relaxing the Universal Quantifier of the Division in Fuzzy Relational Databases", *International Journal of Intelligent Systems*, Vol. 16-6, (2001), pp 713-742.

[4] Galindo J., Medina J.M., Pons O., Cubero J.C., A Server for Fuzzy SQL Queries, "Flexible Query Answering Systems", eds. T. Andreasen, H. Christiansen and H.L. Larsen, *Lecture Notes in Artificial Intelligence (LNAI)*, 1495, Ed. Springer,(1998), pp. 164-174.

[5] Litwin W. and Abdellatif A., "Multidatabase Interoperabilty", *IEEE Computer journal*, (19)12, (1986), pp 10-18.

[6] Litwin, W., Abdellatif, A., Zeroual, A., Nicolas, B. "MSQL: A multidatabase language", *Information sciences*, 49, (1989), pp 59-101.

[7] Lee, Jeong-Oog, Baik, Doo-Kwon, "SemQL: a semantic query language for multidatabase systems",In: *Proceedings of the eighth international conference on Information and knowledge management*, United States, (1999), pp 259-266.

[8] Lim Ee Peng, Chiang Roger H.L., Cao Yinyan "Tuple source relational model: A source aware data model for multidatabase", *Data and Knowledge Engineering*, 29, (1999), pp 83-114.

[9] Litwin W. et al, "SIRIUS Systems for Distributed Data Management", In: Schneider, H.J., ed., *Distributed Databases*, New Yark: North-Holland, 1982.

[10] Juan C. Lavariega, Susan D. Urban, "An Object Algebra Approach to Multidatabase Query Decomposition in Donaj", *Distributed and Parallel Databases*, Volume 12, Issue 1, (2002), 27-71.

[11] Rundensteiner, E.A., Hawkes, L.W. and Bandler, W., "On Nearness Measures in Fuzzy Relational Data Models", *International Journal of Approximate Reasoning*, (3), (1989), 267-298.

[12] Sharma A.K., Goswami A., Gupta D.K., "FTS Relational Data Model for Source-aware Fuzzy Relational Multidatabases", In International Conf. *Proc. of ISCC'04*, Alexandria, Egypt, (2004), pp 134-139.