Relational Languages for Metadata Integration

CATHARINE M. WYSS and EDWARD L. ROBERTSON
Indiana University

In this article, we develop a relational algebra for metadata integration, *Federated Interoperable Relational Algebra* (FIRA). FIRA has many desirable properties such as compositionality, closure, a deterministic semantics, a modest complexity, support for nested queries, a subalgebra equivalent to canonical Relational Algebra (RA), and robustness under certain classes of schema evolution. Beyond this, FIRA queries are capable of producing fully dynamic output schemas, where the number of relations and/or the number of columns in relations of the output varies dynamically with the input instance. Among existing query languages for relational metadata integration, only FIRA provides generalized dynamic output schemas, where the values in any (fixed) number of input columns can determine output schemas.

Further contributions of this article include development of an extended relational model for metadata integration, the *Federated Relational Data Model*, which is strictly downward compatible with the relational model. Additionally, we define the notion of *Transformational Completeness* for relational query languages and postulate FIRA as a canonical transformationally complete language. We also give a declarative, SQL-like query language that is equivalent to FIRA, called *Federated Interoperable Structured Query Language* (FISQL).

While our main contributions are conceptual, the federated model, FISQL/FIRA, and the notion of transformational completeness nevertheless have important applications to data integration and OLAP. In addition to summarizing these applications, we illustrate the use of FIRA to optimize FISQL queries using rule-based transformations that directly parallel their canonical relational counterparts. We conclude the article with an extended discussion of related work as well as an indication of current and future work on FISQL/FIRA.

Categories and Subject Descriptors: H.2.1 [**Database Management**]: Logical Design—*Data models*; H.2.3 [**Database Management**]: Languages—*Query languages*

General Terms: Languages

Additional Key Words and Phrases: Data integration, federated databases, federated data model, interoperability, metadata integration, metadata querying, multidatabases, relational query algebra, schema integration, transformational completeness

The work of E. L. Robertson was supported in part by National Science foundation (NSF) grant IIS-82407.

Authors' address: Indiana University, Computer Science Department, Bloomington, IN 47405; email: {cmw,edrbtsn}@cs.indiana.edu.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 1515 Broadway, New York, NY 10036 USA, fax: +1 (212) 869-0481, or permissions@acm.org.
© 2005 ACM 0362-5915/05/0600-0624 $5.00

1. INTRODUCTION

In this article, we present a formal paradigm for transformationally complete relational metadata integration. Our paradigm consists of an extended relational algebra, *Federated Interoperable Relational Algebra* (FIRA). FIRA augments canonical Relational Algebra (RA) with the ability to query and restructure metadata along with data directly within the relational model. Furthermore, FIRA has an equivalent SQL-like counterpart, called *Federated Interoperable Structured Query Language* (FISQL). While our main contributions are conceptual, the FISQL/FIRA framework nevertheless has important applications to data integration and OLAP, which we now summarize.

The field of data integration has interested database researchers for decades. The dominant architectural model today is *federated*, where sources export (import) views to (from) a mediated schema. This requires both *wrappers* encapsulating source data repositories as well as *mapping functions* giving the translation between source data and/or schemas and the mediated schema. Wrappers, mapping functions, and mediated schemas are crucial concepts in data integration, whether the overall network architecture is centralized, peer-to-peer, or hybrid [Halevy 2004].

Our language, FIRA, can assist with the creation and maintenance of wrappers, mediating functions, and mediated schemas, especially in the case of relational data sources. FIRA is more robust under schema evolution than traditional relational languages, as several examples throughout the paper illustrate. Furthermore, one open problem that FIRA addresses in data integration is that of dynamically varying schemas, where target schemas may depend dynamically on source data. For instance, relational "pivot"-type operations require certain data values to become attribute names. An example of this is the translation between Indianapolis and Chicago stores in our sales federation (Figure 1). FIRA extends contemporary solutions to this problem with more generality. For example, dynamic transformations are no longer restricted to a single column per query as in SchemaSQL [Gingras and Lakshmanan 1998; Lakshmanan et al. 2001]. Also, the federated data model underpinning FIRA provides for a clean extension of RA to federations without recourse to a two-tiered language as in MD-SQL [Rood et al. 1999] or MQL/MA [Wyss and Van Gucht 2001; Wyss et al. 2001]. A deeper discussion of related work appears in Section 6.

Another important application area for FIRA is OnLine Analytical Processing (OLAP). Many relational transformations required in OLAP contain a "relational transpose" similar to our τ operator (Section 3.8). However, the semantics of this operation within the constraints of the relational model have not been well understood. FIRA leads to important theoretical results concerning relational transposition, which we report elsewhere [Robertson and Wyss 2004]. Furthermore, we are working to extend FISQL with aggregation capabilities to better facilitate OLAP applications within the FISQL/FIRA framework.

1.1 Motivating Example

As an example motivating our languages, consider the relational tables shown in Figure 1. In Figure 1, sales data for Amart stores in three midwestern

Indy

Sales

Store	Dept	AvgSales
PineSt	Wmn	62500
WestRd	Wmn	75000
AndAve	Wmn	81500
PineSt	Men	50000
AndAve	Men	73500
PineSt	Toddler	41250
WestRd	Toddler	55000
AndAve	Toddler	68500

Chicago

AvgSales

Store	Wmn	Men	Boy	Girl
CedarRd	48500	35000	25500	$\perp$
CtrSq	55500	50000	32000	52500
WashSt	63500	58500	42250	58500
IllSt	78000	63250	50000	65500

Milw

LincAveSales

Dept	AvgSales	LCode
Girl	45000	1
Boy	55000	2
Men	65000	3
Wmn	80000	4

FreePkSales

Dept	AvgSales	FCode
Girl	35000	A
Men	48500	B
Wmn	55000	C

WashStSales

Dept	AvgSales	WCode
Boy	28500	B38
Men	46500	C18
Wmn	60000	X27

Fig. 1. Average sales for Amart stores in three midwestern cities.

```
SELECT I.Dept, I.Store
FROM Indy.Sales AS I, Chicago.AvgSales AS C
WHERE (I.Dept = "Wmn" AND C.Wmn > I.AvgSales)
     OR (I.Dept = "Men" AND C.Men > I.AvgSales)
```

Fig. 2. Using SQL to compare Amart data.

cities is shown. Each city uses a different relational representation of their sales data. The goal is to facilitate posing queries comparing data in the three cities. An SQL query comparing the Indianapolis and Chicago data is shown in Figure 2. This query returns pairs of Indianapolis departments and stores that make less than at least one comparable department in some Chicago store.

The SQL for this comparison has several problems, centering around the fact that the compared departments are "hard-coded" into the query. Thus, for example, if any Chicago store begins offering clothing for toddlers, this query will no longer perform the correct comparisons. Similarly, if any Indianapolis stores change their department structure, the query is broken. Furthermore, this approach does not scale well as the number of departments increases. For 50 or 100 departments, the query would be daunting to write. In contrast, compare the concise representation in FISQL which is independent of the specific departments (Figure 10, Section 4.1). Previous work has coined the term *Schema Independence* for queries that are robust under this type of schema evolution [Lakshmanan et al. 2001; Masermann and Vossen 2000].

Another class of problems with the SQL query centers around the problem of missing or incomplete information. If Chicago stores decide to discontinue men's clothing, the query in Figure 2 will break because "`C.Men`" no longer refers to a legitimate column in `Chicago.AvgSales`. In contrast, FISQL assumes a

federated data model which differs from the relational model in its approach to missing information, and behaves gracefully under such losses of information. At the same time, the federated model is strictly downward compatible with the relational model, in that every relation has a unique federated counterpart (see Definition 3.5, Section 3.2). Furthermore, this counterpart can be generated purely syntactically without recourse to semantic information about the data (unlike the transformation to the tabular model in Lakshmanan et al. [1999], for example).

1.2 Main Contributions

In summary, the main contributions of the current article are as follows:

- We introduce the *Federated Relational Model* in Section 2.1 as a straightforward syntactic extension of the relational model. The federated model treats relation and attribute names as first class domain elements, and behaves well under incomplete information.
- We introduce the notion of *Transformational Completeness* for query languages in Section 2.2. This notion captures our intuition concerning the completeness of a query language for relational data $\leftrightarrow$ metadata transformations.
- Our foremost contribution in this article is to introduce the algebraic query language FIRA, which is our paradigm for transformationally complete relational metadata integration. FIRA has several advantageous properties beyond current relational metadata integration languages, including:
  - *Compositionality*—A FIRA query maps a set of federated databases to a single federated database. This closure is a natural parallel of RA closure, since RA accepts a finite number of relations and returns a single relation as output.
  - *Downward Compatibility*—FIRA contains a subalgebra isomorphic to the canonical RA. This entails that RA equivalences translate directly to FIRA equivalences. Thus (for example), FIRA Cartesian Product is commutative and associative, unlike previous algebras [Grant et al. 1993].
  - *Generality*—FIRA allows dynamic schema transformations in more generality than previous languages such as SchemaSQL [Lakshmanan et al. 2001] where only a single column of data could be promoted to metadata per query.
  - *Nested Queries*—FIRA includes support for nested queries, and FIRA queries can be arbitrarily chained together in finite sequences. This is in contrast to previous languages that have used view mechanisms to integrate data and metadata, such as SchemaSQL [Lakshmanan et al. 2001].
- In Section 4, we present FISQL and show subsequently that it is equivalent to FIRA (Section 4.4). The FISQL-FIRA equivalence parallels the SQL-RA equivalence, which means rule-based query optimization techniques developed for SQL immediately carry over to FISQL, as we show in Section 5.

—Finally, in Section 6, we give a targeted overview of related work, comparing properties of previous relational metadata integration languages to FISQL/FIRA. A tabular summary of the survey is given in Table I.

In the next section, we begin by detailing formal notation and concepts assumed throughout the presentation.

2. FORMAL PRELIMINARIES

For the remainder of the article, we assume familiarity with the standard relational model and Relational Algebra (RA). To fix these notions, we briefly recapitulate what we understand to be accepted definitions of these concepts.

We assume a domain of atomic elements, denoted ***dom***, which contains alphanumeric strings, such as `123`, `abc` or `SomeElement`. We use teletype font to denote specific elements of ***dom***. In addition, we use a special element, $\perp$, called the *null* element. We assume $\perp \notin$ ***dom***, so that $\perp$ can be used to distinguish noninformation within the model.

For metadata, we also assume a domain of atomic elements, denoted $\mathfrak{M}_0$, which is usually considered to contain alphanumeric strings beginning with a letter. We assume $\mathfrak{M}_0 \subset$ ***dom***.[1]

Definition 2.1

(1) A *(Canonical) Tuple*, t is a mapping from a finite set $S \subset \mathfrak{M}_0$ to ***dom*** $\cup \{\perp\}$. Elements of S are termed *attributes*. The square-bracket notation $t[A]$ is used to signify the element $t(A)$ for $A \in S$.
(2) A *(Canonical) Relation* has a name $N \in \mathfrak{M}_0$ and a finite schema $S \subset \mathfrak{M}_0$. The relation body consists of a finite set of (canonical) tuples $t : S \rightarrow$ ***dom*** $\cup \{\perp\}$.
(3) A *(Canonical) Database* consists of a finite set of (canonical) relations.

Definition 2.2

(1) The *Relational Algebra* (RA) is understood to be the query language generated by applying the six relational operations ρ (Renaming), σ (Selection), π (Projection), $\times$ (Cartesian Product), $\cup$ (Set Union), and—(Set Difference). A query in the RA maps a set of input relations (i.e., a database) to a single output relation.
(2) A query language that can express all queries of the RA is said to be *Relationally Complete* [Codd 1970].

The notion of Relational Completeness is often used to gauge expressivity within the relational model. Beyond this, the data complexity of the RA is LOGSPACE. This is often used as a second gauge for relational query languages, since efficient query processing is of paramount importance in large data sets. The LOGSPACE class is considered to contain enough power for most common query tasks, while remaining efficient in terms of execution times. The RA itself is particularly suited to optimized query execution, since several nice algebraic

[1]Classically, relational metadata and data are not compared within the model. However, relational metadata are represented as alphanumeric strings beginning with a letter, so in principle there is no problem with comparing metadata and data literals (as strings).

rules give rise to simple but effective rule-based query optimization strategies [Garcia-Molina et al. 2000].

2.1 The Federated Relational Data Model

One of the main contributions of our work is to extend the relational model to incorporate metadata. The resulting data model is termed the *Federated Relational Data Model*, or simply the *Federated Model*. In the federated model, relational metadata can be compared and transformed along with relational data. However, the meta-ness of metadata in the ordinary relational model must be recapitulated in the federated model, only at the "next-higher" level. Thus, we assume a countable domain of *meta-metadata*, denoted $\mathfrak{M}_1$, containing elements distinct from any in $\boldsymbol{dom}$ and $\mathfrak{M}_0$; that is, $\mathfrak{M}_1 \cap \boldsymbol{dom} = \emptyset$ and $\mathfrak{M}_1 \cap \mathfrak{M}_0 = \emptyset$. In the federated model, relational data and metadata are allowed to occupy both data and schema positions.[2] However, federated metadata (elements of $\mathfrak{M}_1$) are not allowed as data.

Definition 2.3

(1) A *(Federated) Tuple* is a mapping from a finite set $S \subset \boldsymbol{dom} \cup \mathfrak{M}_1$ to $\boldsymbol{dom} \cup \{\bot\}$. S is known as the *Schema* of the tuple, that is, $S = \mathit{schema}(t)$.
(2) A *(Federated) Relation* has a name $N \in \boldsymbol{dom}$. The relation body consists of a finite set of (federated) tuples.
(3) A *(Federated) Database* has a name $D \in \mathfrak{M}_0$. The database body consists of a finite set of (federated) relations.
(4) A *Federation* consists of a finite set of (federated) databases.

Definition 2.4

(1) A federated relation can be seen as a pair $\langle N, B\rangle$, where $N \in \boldsymbol{dom}$ and B is a finite set of federated tuples. Given federated relation $\mathcal{R} = \langle N, B\rangle$, we use the notation *name*($\mathcal{R}$) and *body*($\mathcal{R}$) to refer abstractly to N and B (respectively).
(2) Given a federated relation $\mathcal{R}$, we define its *Schema* to be

$$\mathit{schema}(\mathcal{R}) = \bigcup_{t \in \mathit{body}(\mathcal{R})} \mathit{schema}(t).$$

Note that a federated relation is allowed to contain tuples with (possibly) differing schemas. The schema of the relation is the union of the schemas of tuples it contains, so that the schema of a federated relation depends critically on the relation *instance*. This is an important design decision in the creation of the federated model, and facilitates dynamic schema transformations, such as relational transposition (Section 3.8) or attribute dereference (Section 3.5). One consequence of this decision is that algebraic operations that remove or change tuples may modify the relation schema. As per Definition 2.4, we denote

[2]It may seem odd to allow numbers as attributes and relation names. However, there is no reason in principle to rule this out; in fact, dates or ages (for example) may be quite useful column or relation names. In any case, the formalism for disallowing these cases is unwieldy, so for clarity we defer such considerations to implementation.

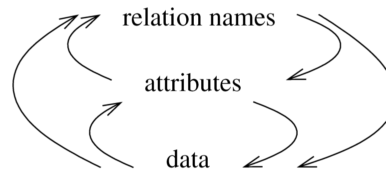

Fig. 3. Data-metadata transformations.

the schema of a federated relation abstractly as *schema*($\mathcal{R}$), even though the extension of this set varies with the relation instance. This view is notationally convenient, and we use it throughout the remainder of this article.

2.2 Transformational Completeness

So far, we have seen federated analogs of the relational notions of tuple, relation, database, and schema. Another analogous concept is that of *Transformational Completeness*. We say that a query language is *Transformationally Complete* if (i) it is relationally complete and (ii) it can express schema-independent queries encompassing all six data-metadata transformations depicted in Figure 3. Note that canonical RA is not transformationally complete in this sense, as previous work has indicated [Lakshmanan et al. 2001].

Several languages have been proposed that capture some or all of the transformations depicted in Figure 3. These languages are discussed in more detail in Section 6 after we present our transformationally complete query languages FIRA and FISQL (Sections 3 and 4, respectively). In fact, one of our contributions is to formalize the notion of "transformational completeness" by postulating FIRA as a paradigmatic transformationally complete language. This is not to say FIRA is the final word in relational metadata integration; rather we use FIRA as a formal archetype for what it means to be transformationally complete (in the same way as RA is a formal archetype for what it means to be relationally complete). Thus, transformational completeness and FIRA are the federated analog of relational completeness and RA.

Note that the transformations of Figure 3 must be carried out in a manner that preserves the semantics of the underlying data. For the relational model, one formalization of this type of symmetry is encoded in the notion of *BP-Completeness* [Abiteboul et al. 1995]. To fully capture the notion of symmetry inherent in federated transformations, ongoing work is extending BP-Completeness and related notions to the federated data model.

2.3 Higher-Order Federated Data Models

In principle, there is no reason to stop with federations and $\mathfrak{M}_1$. For $k \geq 2$ we can assume a "higher-order" metadata domain $\mathfrak{M}_k$ that is disjoint from ***dom*** and from $\mathfrak{M}_j$ where $j < k$. We can then define the concepts of tuple, relation, database, federation, set of federations, set of sets of federations, etc., and associated notions. The query languages for these "higher-order" relational models will have affinities with higher-order logics, just as FIRA and FISQL (and similar languages) have affinities with second-order logic. Ongoing work is characterizing the resulting relational hierarchy and languages. In particular,

extended federated models provide a segue to tree-structured databases and XML.

In the present federated model, elements of $\mathfrak{M}_1$ cannot be demoted to data within the federated model; thus the "chain of removal" ends with $\mathfrak{M}_1$.

3. FEDERATED INTEROPERABLE RA

In this section, we present the query algebra called *Federated Interoperable Relational Algebra*. Each operator in FIRA accepts a fixed number of databases as input (i.e., a federation) and returns a single database. This closure is a natural parallel of RA closure, since RA accepts a fixed number of *relations* and returns a single *relation* as output. As with RA, some operators are parameterized by elements of the input schema.

3.1 Basic Terms

Basic terms in FIRA are database names or database variables. For example, "`Indy`" is a basic term in the FIRA. We use variables of the form $\mathcal{D}_1, \mathcal{D}_2, \ldots$ to denote databases.

3.2 The Relational Core of FIRA

Each of the six RA operators $(\rho, \sigma, \pi, \times, \cup, -)$ has a federated counterpart within FIRA. We use hatted notation for these FIRA counterparts ($\hat{\rho}$, $\hat{\sigma}$, etc.). The definitions of the FIRA relational algebra operators include applications of canonical RA operators to federated relation bodies. Given federated relation $\mathcal{R}$, $body(\mathcal{R})$ is not, strictly speaking, a canonical relation. However, these notions are so close that we ignore the difference to facilitate concise definitions.[3]

One twist in the definition of the federated counterparts for unary RA operators is that we need to allow relation renaming in addition to attribute renaming; this is done in Definition 3.1(1). Selection and Projection are straightforward (Definitions, 3.1(2) and 3.1(3), respectively).

Definition 3.1 (*Federated Unary Relational Operators*)

(1) (*Renaming*) Let $\mathcal{D}$ be a federated database. There are two cases.
   (a) (*General Renaming*) Let $A_i, B_i \in \boldsymbol{dom} \cup \mathfrak{M}_1$ for $1 \leq i \leq n$. Then

$$\hat{\rho}_{A_1 \to B_1, \ldots, A_n \to B_n}(\mathcal{D}) = \{\langle name(\mathcal{R}), \rho_{A_1 \to B_1, \ldots, A_n \to B_n}(body(\mathcal{R}))\rangle | \mathcal{R} \in \mathcal{D}\}.$$

   (b) (*Relation Specific Renaming*) Let $A_i, B_i \in \boldsymbol{dom} \cup \mathfrak{M}_1$ for $1 \leq i \leq n$ and $N, M \in \boldsymbol{dom}$. Then

$$\hat{\rho}^{N \to M}_{A_1 \to B_1, \ldots, A_n \to B_n}(\mathcal{D}) = \{\langle M, \rho_{A_1 \to B_1, \ldots, A_n \to B_n}(body(\mathcal{R}))\rangle | \mathcal{R} \in \mathcal{D}, name(\mathcal{R}) = N\} \cup \{\mathcal{R} \in \mathcal{D} | name(\mathcal{R}) \neq N\}.$$

[3]Formally, for federated relation $\mathcal{R}$, we consider $schema(\mathcal{R})$ as the set "S" in Definition 2.1, extending tuples in $body(\mathcal{R})$ with null values where necessary. Since RA operators do not demote relational metadata to data, we can view elements of $\boldsymbol{dom}$ and $\mathfrak{M}_1$ in $schema(\mathcal{R})$ as canonical metadata constants so that the action of RA operators on columns headed by these elements is the same as for columns headed by elements of $\mathfrak{M}_0$.

(2) (*Selection*) Let $\mathcal{D}$ be a federated database and C be a well-formed Boolean selection condition. Then

$$\hat{\sigma}_C(\mathcal{D}) = \{\langle name(\mathcal{R}), \sigma_C(body(\mathcal{R}))\rangle | \mathcal{R} \in \mathcal{D}\}.$$

(3) (*Projection*) Let $\mathcal{D}$ be a federated database and $A_k \in \textbf{\textit{dom}} \cup \mathfrak{M}_1$ for $1 \leq k \leq n$. Then

$$\hat{\pi}_{A_1,\ldots,A_n}(\mathcal{D}) = \{\langle name(\mathcal{R}), \pi_{A_1,\ldots,A_n}(body(\mathcal{R})) | \mathcal{R} \in \mathcal{D}\}.$$

For the binary RA operators, the fact that relation names are now a part of the formalism means that we must be careful when defining the federated counterparts of these operations. When operating on federated databases, the FIRA counterparts of the binary RA operators behave consistently with the RA by acting non-trivially only on *like named relations*. Furthermore, since schemas vary dynamically in FIRA, we assume *outer* versions of the canonical RA operators $\times$, $\cup$, and $-$, so that mismatched schemas will still provide sensible answers.

Definition 3.2 (*Federated Cartesian Product*). Let $\mathcal{D}_1$ and $\mathcal{D}_2$ denote federated databases. Their *Federated Cartesian Product* is defined as

$$\mathcal{D}_1 \hat{\times} \mathcal{D}_2 = \{\langle name(\mathcal{R}_1), body(\mathcal{R}_1) \times body(\mathcal{R}_2)\rangle \mid \mathcal{R}_1 \in \mathcal{D}_1, \mathcal{R}_2 \in \mathcal{D}_2 \text{ and } name(\mathcal{R}_1) = name(\mathcal{R}_2)\}.$$

Note that unmatched relations in either database are dropped in the product, so the cardinality of $\mathcal{D}_1 \hat{\times} \mathcal{D}_2$ is equal to or less than that of $\mathcal{D}_1$. This decision may seem odd, but has two important consequences. First, $\hat{\times}$ is commutative and associative, unlike in previous algebras that define the Cartesian Product of databases as all pairwise combinations of products [Grant et al. 1993]. Also, there exists an embedding of canonical relations into federated relations that preserves RA as a subalgebra of FIRA (Theorem 3.6, below). For similar reasons, federated set union and set difference are defined analogously, as follows.

Definition 3.3 (*Federated Set Union*). Let $\mathcal{D}_1$ and $\mathcal{D}_2$ denote federated databases. Their *Federated Set Union* is defined as

$$\begin{aligned}\mathcal{D}_1 \hat{\cup} \mathcal{D}_2 = \{&\langle name(\mathcal{R}_1), body(\mathcal{R}_1) \cup body(\mathcal{R}_2)\rangle \\ &\mid \mathcal{R}_1 \in \mathcal{D}_1, \mathcal{R}_2 \in \mathcal{D}_2 \text{ and } name(\mathcal{R}_1) = name\mathcal{R}_2\} \\ &\bigcup \{\mathcal{R}_1 \in \mathcal{D}_1 | \text{ there is no } \mathcal{R}_2 \in \mathcal{D}_2 \text{ such that } name(\mathcal{R}_1) = name\mathcal{R}_2\} \\ &\bigcup \{\mathcal{R}_2 \in \mathcal{D}_2 | \text{ there is no } \mathcal{R}_1 \in \mathcal{D}_1 \text{ such that } name(\mathcal{R}_2) = name\mathcal{R}_1\}.\end{aligned}$$

Definition 3.4 (*Federated Set Difference*). Let $\mathcal{D}_1$ and $\mathcal{D}_2$ denote federated databases. The *Federated Set Difference* of $\mathcal{D}_1$ and $\mathcal{D}_2$ is defined as

$$\begin{aligned}\mathcal{D}_1 \hat{-} \mathcal{D}_2 = \{&\langle name(\mathcal{R}_1), body(\mathcal{R}_1) - body(\mathcal{R}_2)\rangle \\ &\mid \mathcal{R}_1 \in \mathcal{D}_1, \mathcal{R}_2 \in \mathcal{D}_2 \text{ and } name(\mathcal{R}_1) = name\mathcal{R}_2\} \\ &\bigcup \{\mathcal{R}_1 \in \mathcal{D}_1 | \text{ there is no } \mathcal{R}_2 \in \mathcal{D}_2 \text{ such that } name(\mathcal{R}_1) = name\mathcal{R}_2\}.\end{aligned}$$

The six FIRA operators $\hat{\rho}$, $\hat{\sigma}$, $\hat{\pi}$, $\hat{\times}$, $\hat{\cup}$, and $\hat{-}$ are in and of themselves a compositionally closed algebra on federated databases. This algebra is thus

a subalgebra of full FIRA (which we have yet to define), and furthermore is equivalent to canonical RA. To see this, we require a well-defined embedding of canonical relations into federated databases. Note that since $\boldsymbol{dom} \subset \boldsymbol{dom} \cup \mathfrak{M}_1$, we can view a canonical relation as the body of a federated relation without issue. Let ε be a distinguished symbol in $\boldsymbol{dom}$, called the "empty name". We federate canonical relations simply by assigning them the name ε.

Definition 3.5. Let R be a canonical relation. The *Federated Counterpart of* R, denoted *federate*(R), is the federated database $\{\langle \varepsilon, R\rangle\}$.

THEOREM 3.6. *RA is isomorphic to a subalgebra of FIRA.*

PROOF. Definition 3.5 defines a mapping, *federate*, from canonical relations into federated databases given by $R \mapsto$ *federate*(R). It is easy to see that *federate* is injective. Furthermore, *federate* is a *homomorphism* with respect to the RA operators and their FIRA counterparts so that, for example, *federate*$(\sigma_C(R)) = \hat{\sigma}_C($*federate*$(R))$, *federate*$(R \cup S) =$ *federate*$(R) \hat{\cup}$ *federate*(S), etc.

On the other hand, consider the class $\mathfrak{D}$ that consists of exactly the federated databases that are the image of some canonical relation under *federate*. Clearly, *federate* is surjective for this class. It is relatively easy to see that we have closure under the FIRA operations $\hat{\rho}$, $\hat{\sigma}$, $\hat{\pi}$, $\hat{\times}$, $\hat{\cup}$, and $\hat{-}$ within $\mathfrak{D}$.

Thus, *federate* induces a subalgebra of FIRA that is isomorphic to RA. We term this subalgebra the *Relational Core* of FIRA. □

Theorem 3.6 is important because it precisely characterizes how FIRA semantics properly extends RA semantics. Previous algebras have not shown such a result.

The next subsections introduce the operators of full FIRA that move beyond the relational core.

3.3 Drop Projection

FIRA includes a projection operator that allows one to remove elements from relation schemas, denoted $\pi\!\!\!\!\!\,\,/$.

Definition 3.7

(1) (*Drop Projection for Federated Relations*) Let $\mathcal{R}$ be a federated relation and $A \in \boldsymbol{dom} \cup \mathfrak{M}_1$. Then

$$\not\pi_A(\mathcal{R}) = \{\langle name(\mathcal{R}), \pi_{schema(\mathcal{R})-A}(body(\mathcal{R})\rangle\}.$$

(2) (*Drop Projection for Federated Databases*) Let $\mathcal{D}$ be a federated database and $A \in \boldsymbol{dom} \cup \mathfrak{M}_1$. Then $\not\pi_A(\mathcal{D}) = \{\not\pi_A(\mathcal{R}) | \mathcal{R} \in \mathcal{D}\}$. In addition, we use the shorthand notation $\hat{\not\pi}_{A_1,\ldots,A_n}(\mathcal{D})$ for $A_k \in \boldsymbol{dom} \cup \mathfrak{M}_1$ $(1 \leq k \leq n)$ to mean $\hat{\not\pi}_{A_1}(\cdots(\hat{\not\pi}_{A_n}(\mathcal{D}))\cdots)$.

Note that since federated schemas vary dynamically with instances, the drop projection cannot uniformly be imitated with canonical projection. In fact, recent work shows that drop projection is not superfluous for polymorphic versions of the canonical RA [Van den Bussche and Waller 2002].

$\mathfrak{r}_3$	$\mathfrak{a}_3$	Dept	AvgSal	FCode
FreePkSales	Dept	Girl	35000	A
FreePkSales	Dept	Men	48500	B
FreePkSales	Dept	Wmn	55000	C
FreePkSales	AvgSales	Girl	35000	A
FreePkSales	AvgSales	Men	48500	B
FreePkSales	AvgSales	Wmn	55000	C
FreePkSales	FCode	Girl	35000	A
FreePkSales	FCode	Men	48500	B
FreePkSales	FCode	Wmn	55000	C

Fig. 4. The relation $\downarrow_3$ (`Milw.FreePkSales`).

3.4 The Down Operator

FIRA contains a family of operators which explicitly demote relational metadata to federated data. Notationally, we use fixed constants from $\mathfrak{M}_1$ to signify columns created to hold relational metadata. We will assume two families of constants in $\mathfrak{M}_1$ which we represent using lowercase, subscripted versions of the "Fraktur" font, such as $\mathfrak{r}_3$, and $\mathfrak{a}_{21}$ (for relation, and attribute names, respectively). Note that these constants can appear in federated relation schemas, however they cannot be demoted to federated data. This is a consequence of "stopping" the relational hierarchy at federations (as discussed in Section 2.3).

Definition 3.8 (*Down Operators*)

(1) (*Down Operators for Federated Relations*) Let $\mathcal{R}$ be a federated relation and $i \in \mathbb{N}$ be a fixed natural number.

(a) Let $name(\mathcal{R}) = N \in \boldsymbol{dom}$ and $schema(\mathcal{R}) \cap \boldsymbol{dom} = \{A_1, \ldots, A_n\}$. We define $metadata_i(\mathcal{R})$ to be the following set of federated tuples:

$\mathfrak{r}_i$	$\mathfrak{a}_i$
N	A_1
N	A_2
$\vdots$	$\vdots$
N	A_n

(b) The *down of* $\mathcal{R}$ *with respect to* i, denoted $\downarrow_i (\mathcal{R})$ is the federated relation

$$\downarrow_i (\mathcal{R}) = \langle name(\mathcal{R}), metadata_i(\mathcal{R}) \times \pi\!\!\!/_{\mathfrak{r}_i, \mathfrak{a}_i}(body(\mathcal{R}))\rangle.$$

(2) (*Down Operators for Federated Databases*) Let $\mathcal{D}$ be a federated database. Then

$$\downarrow_i (\mathcal{D}) = \{\downarrow_i (\mathcal{R}) | \mathcal{R} \in \mathcal{D}\}.$$

Example 3.9. The federated relation $\downarrow_3$(`Milw.FreePkSales`) is shown in Figure 4.

Note that i only affects the choice of constants from $\mathfrak{M}_1$ labeling the output columns of $metadata_i(\mathcal{R})$. The drop projection in the body of $\downarrow_i (\mathcal{R})$ ensures that we only add meta-metadata subscripted by i *once*. In other words,

$\downarrow_i(\downarrow_i(\mathcal{R})) = \downarrow_i(\mathcal{R})$. This plus the fact that we do not include meta-metadata as data in the relation $metadata_i(\mathcal{R})$ means that $\downarrow_i (\downarrow_j (\mathcal{R})) = \downarrow_i (\mathcal{R}) \bowtie \downarrow_j (\mathcal{R})$ for $i, j \in \mathbb{N}$.

3.5 Attribute Dereference

FIRA includes an operator for projecting a relation R on attributes determined by cell values in another attribute. For each tuple $t \in R$, instead of projecting that tuple on field A, we project on the field named by $t[A]$. This operation is termed *Attribute Dereference* and is denoted Δ.

Definition 3.10 (*Attribute Dereference*)

(1) (*Attribute Dereference for Federated Relations*) Let $\mathcal{R}$ be a federated relation and $A, B \in \boldsymbol{dom} \cup \mathfrak{M}_1$. Then $\Delta_A^B(\mathcal{R}) = \langle name(R), R' \rangle$ where R' is obtained from $body(\mathcal{R})$ tuple-by-tuple as follows: For $t \in body(\mathcal{R})$, we obtain $s \in R'$ as:

$$s[X] = \begin{cases} t[t[A]] \text{ iff } X = B; \\ t[X] \text{ otherwise.} \end{cases}$$

(2) (*Attribute Dereference for Federated Databases*) Let $\mathcal{D}$ be a database and $A, B \in \boldsymbol{dom} \cup \mathfrak{M}_1$. Then $\Delta_A^B(\mathcal{D}) = \{\Delta_A^B(\mathcal{R}) | \mathcal{R} \in \mathcal{D}\}$.

Example 3.11. Consider dereferencing `Chi.AvgSales` by the Indy departments. We will first augment `Chi.AvgSales` with the column of Indy department names (Figure 5(a)); then we can dereference the resulting relation to obtain the desired information (Figure 5(b)).

Dereferencing is common when comparing data from relations having "transposed" structure, such as the `Chi.AvgSales` and the `Indy.Sales` relations. This type of structural mismatch is commonly found when comparing spreadsheet data to native relational data. In these cases, dereference terms will most often be included within selections and projections. For these situations, we provide convenient notational shorthands.

Definition 3.12

(1) Let $\mathcal{R}$ be a federated relation and $A, B \in \boldsymbol{dom} \cup \mathfrak{M}_1$. Let $op \in \{<, >, =, \neq, \leq, \geq\}$. Then $\sigma_{\overline{A}\ op\ B}(\mathcal{R})$ is shorthand notation for

$$\varkappa_X(\sigma_{X\ op\ B}(\Delta_A^X(\mathcal{R}))).$$

where $X \in (\boldsymbol{dom} \cup \mathfrak{M}_1) - schema(\mathcal{R})$ is arbitrary (but fixed). Similarly, we allow selection terms of the form $\sigma_{A\ op\ \overline{B}}(\mathcal{R})$ or $\sigma_{\overline{A}\ \mathbf{op}\ \overline{B}}(\mathcal{R})$.

(2) Let $\mathcal{R}$ be a federated relation and $A, B \in \boldsymbol{dom} \cup \mathfrak{M}_1$. Then $\pi_{\overline{A}}^B(\mathcal{R})$ is shorthand for $\pi_B(\Delta_A^B(\mathcal{R}))$. When such projection terms are included in a list, it is understood that the dereferencing will take place first, then the projections.

In general, the overline notation ($\overline{A}$) means we dereference on column A first, then perform the desired selection or projection on the result of the dereference.

$\mathcal{R}' = \texttt{Chi.AvgSales} \hat{\times} \rho_{\texttt{Dept}\rightarrow\texttt{IDept}}(\pi_{\texttt{Dept}}(\texttt{Indy.Sales}))$:

IDept	Store	Wmn	Men	Boy	Girl
Wmn	CedarRd	48500	35000	25500	⊥
Wmn	CtrSq	55500	50000	32000	52500
Wmn	WashSt	63500	58500	42250	58500
Wmn	IllSt	78000	63250	50000	65500
Men	CedarRd	48500	35000	25500	⊥
Men	CtrSq	55500	50000	32000	52500
Men	WashSt	63500	58500	42250	58500
Men	IllSt	78000	63250	50000	65500
Toddler	CedarRd	48500	35000	25500	⊥
Toddler	CtrSq	55500	50000	32000	52500
Toddler	WashSt	63500	58500	42250	58500
Toddler	IllSt	78000	63250	50000	65500

(a) Augmenting Chicago Relation with Indianapolis Department Names.

$\Delta^{\texttt{ChiSales}}_{\texttt{IDept}}(\mathcal{R}')$:

ChiSales	IDept	Store	Wmn	Men	Boy	Girl
48500	Wmn	CedarRd	48500	35000	25500	⊥
55500	Wmn	CtrSq	55500	50000	32000	52500
63500	Wmn	WashSt	63500	58500	42250	58500
78000	Wmn	IllSt	78000	63250	50000	65500
35000	Men	CedarRd	48500	35000	25500	⊥
50000	Men	CtrSq	55500	50000	32000	52500
58500	Men	WashSt	63500	58500	42250	58500
63250	Men	IllSt	78000	63250	50000	65500
⊥	Toddler	CedarRd	48500	35000	25500	⊥
⊥	Toddler	CtrSq	55500	50000	32000	52500
⊥	Toddler	WashSt	63500	58500	42250	58500
⊥	Toddler	IllSt	78000	63250	50000	65500

(b) Dereferencing Result of (a) to Pair Chicago Sales with Indianapolis Departments.

Fig. 5. Result of augmenting `Chi.AvgSales` and dereferencing.

3.6 Generalized Union

FIRA contains a *generalized union* operator (denoted Σ) that unions all relations within an input database. The result of Σ is a database containing a single federated relation named ε whose body is the outer union of all those in the input. Thus, Σ effectively combines a database into a single federated relation containing all the information within that database.

Definition 3.13 (*Generalized Union*). Let $\mathcal{D}$ be a federated database. Then

$$\Sigma(\mathcal{D}) = \left\{\left\langle \varepsilon, \bigcup_{\mathcal{R}\in\mathcal{D}} body(\mathcal{R})\right\rangle\right\}.$$

Example 3.14. The result of $\Sigma(\texttt{Milw})$ is shown in Figure 6. This query returns a single relation containing the information from all the Milwaukee relations.

Dept	AvgSales	Lcode	Fcode	Wcode
Girl	45000	1	⊥	⊥
Boy	55000	2	⊥	⊥
Men	65000	3	⊥	⊥
Wmn	80000	4	⊥	⊥
Girl	35000	⊥	A	⊥
Men	48500	⊥	B	⊥
Wmn	55000	⊥	C	⊥
Boy	28500	⊥	⊥	B38
Men	46500	⊥	⊥	C18
Wmn	60000	⊥	⊥	X27

Fig. 6. Result of Σ(`Milw`) operation.

3.7 The Partition Operator

The next FIRA operator is the inverse to Σ. The *partition* operator (denoted $\wp$) splits a given relation into several relations, based on names given in a column of the input relation.

Definition 3.15 (*Partition*)

(1) (*Partition for Federated Relations*) Let $\mathcal{R}$ be a federated relation and $A \in \boldsymbol{dom} \cup \mathfrak{M}_1$. Then $\wp_A(\mathcal{R})$ is the federated database

$$\wp_A(\mathcal{R}) = \{\langle a, \sigma_{A=\text{"}a\text{"}}(body(\mathcal{R}))\rangle \mid \exists\, t \in body(\mathcal{R}) \text{ such that } t[A] = a\}.$$

(2) (*Partition for Federated Databases*) Let $\mathcal{D}$ be a database and $A \in \boldsymbol{dom} \cup \mathfrak{M}_1$. Then

$$\wp_A(\mathcal{D}) = \hat{\bigcup_{\mathcal{R}\in\mathcal{D}}} \wp_A(\mathcal{R}).$$

For notational convenience, it is often necessary to "partition" on a constant. By this we mean a renaming of the input relation(s); so that for a fixed domain element $\mathtt{a} \in \boldsymbol{dom}$,

$$\wp_{\text{"}\mathtt{a}\text{"}}(\mathcal{D}) = \hat{\bigcup_{\mathcal{R}\in\mathcal{D}}} \hat{\rho}^{\,name(\mathcal{R})\rightarrow \mathtt{a}}(\mathcal{R}).$$

Example 3.16. Figure 7 shows the result of the operation $\wp_{\texttt{Store}}$(`Indy.Sales`).

3.8 The Transpose Operator

To be transformationally complete, FIRA must include an operator that allows the promotion of data to attribute names. This operator, denoted τ, is termed *Transpose*, due to affinities with the matrix transpose operation. In spreadsheets, a similar operation is termed *Pivot* [Cunningham et al. 2004]. Unlike a spreadsheet, a relation is transposed tuple-by-tuple, as follows.

Definition 3.17 (*FIRA Transpose*)

(1) (*Transpose for Federated Relations*) Let $\mathcal{R}$ be a federated relation and $A, B \in \boldsymbol{dom} \cup \mathfrak{M}_1$. Then the *transpose of A on B of $\mathcal{R}$*, denoted $\tau_A^B(\mathcal{R})$, is a relation having the same name as $\mathcal{R}$, where each tuple, s, in the body of the output

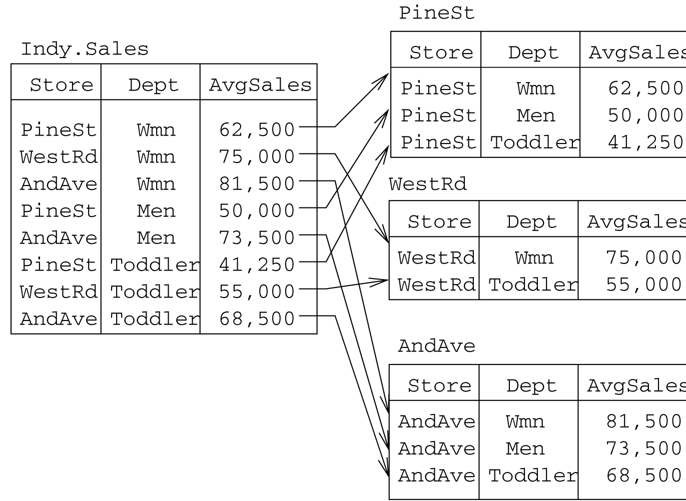

Fig. 7. Partitioning `Indy.Sales`.

Store	Dept	AvgSales	Wmn	Men	Toddler
PineSt	Wmn	62500	62500	⊥	⊥
WestRd	Wmn	75000	75000	⊥	⊥
AndAve	Wmn	81500	81500	⊥	⊥
PineSt	Men	50000	⊥	50000	⊥
AndAve	Men	73500	⊥	73500	⊥
PineSt	Toddler	41250	⊥	⊥	41250
WestRd	Toddler	55000	⊥	⊥	55000
AndAve	Toddler	68500	⊥	⊥	68500

Fig. 8. Result of the operation $\tau^{\texttt{Dept}}_{\texttt{AvgSales}}(\texttt{Indy.Sales})$.

relation is obtained from tuple $t \in body(\mathcal{R})$ as follows.

$$s[X] = \begin{cases} t[A] \text{ iff } X = t[B]; \\ t[X] \text{ iff } X \in schema(t), X \neq t[B]; \\ \bot \text{ otherwise.} \end{cases}$$

(2) (*Transpose for Federated Databases*) Let $\mathcal{D}$ be a database and $A, B \in \boldsymbol{dom} \cup \mathfrak{M}_1$. Then $\tau^B_A(\mathcal{D}) = \{\tau^B_A(\mathcal{R}) | \mathcal{R} \in \mathcal{D}\}$.

Example 3.18. Figure 8 shows the result of the operation $\mathcal{E} := \tau^{\texttt{Dept}}_{\texttt{AvgSales}}$ (`Indy.Sales`). Note that the contents of the `AvgSales` column in Figure 8 are identical with the contents of the column `X` in the dereference operation $\Delta^{\texttt{X}}_{\texttt{Dept}}(\mathcal{E})$. This illustrates that τ and Δ essentially perform inverse operations.

In transposing a relation, null values are explicitly not promoted to column headings. However, in general, many null values appear as data within the new relation. Sometimes it is a simple matter to reduce these null values by "merging" based on key values. In general, for a given relation, finding an equivalent

merged form with a minimal number of tuples is NP-complete [Robertson and Wyss 2004].

FIRA is the query algebra generated by applying finite sequences of the operations $\hat{\rho}$, $\hat{\sigma}$, $\hat{\pi}$, $\hat{\times}$, $\hat{\cup}$, $\hat{-}$, μ, $\downarrow$, Δ, Σ, $\wp$, and τ to federated databases. FIRA is our paradigm for transformationally complete query languages. For usability and familiarity, we provide an equivalent SQL-like query language which is introduced in the next section.

4. FEDERATED INTEROPERABLE SQL

In this section, we present the syntax and semantics of *Federated Interoperable SQL* (FISQL). FISQL is a strict extension of the query sub-language of SQL equivalent to RA (we refer to this sub-language as "core SQL"), in that it contains a sub-language equivalent to core SQL and its semantics are extended in a manner that is wholly consistent with SQL.

Syntactically, the most obvious additions to SQL are the declaration of *metavariables* in the FROM clause and the ability to give parameterized names to output schemas. Semantically, FISQL maps an input federation to an output database whose schema may depend on the input data.

Metavariables in the FROM clause range over relation and attribute names. They allow the creation of queries that have a second-order appearance; however, FISQL has a first-order semantics since metavariables range over metadata as domain elements as opposed to relations themselves. In this, FISQL follows in the tradition of syntactically higher-order query languages with a first-order semantics, such as HiLog [Chen et al. 1990b] and SchemaSQL [Lakshmanan et al. 2001].

Unlike previous languages, FISQL provides explicit keywords signaling dynamically varying output schemas in the SELECT clause, namely the "ON" and "INTO" keywords. This design decision has two main purposes.

First, it gives FISQL downward compatibility with the CREATE VIEW construct of ordinary SQL. In contrast (for example), SchemaSQL uses the CREATE VIEW construct to determine when an otherwise non-dynamic query gives a dynamic result; thus, the same SchemaSQL base query has different semantics if it is contained in a view.[4] This is problematic for the compositionality of the language as well as compatibility with SQL.

Secondly, from a syntactic viewpoint, previous languages such as SchemaLog [Lakshmanan et al. 1997] and SchemaSQL [Lakshmanan et al. 2001] have used a notation whereby a single parameter may unify with a domain element *or* an entire list in the final output. Instead, the FISQL "ON" keyword specifically denotes the particular case of unifying with a list of output values. We find this notation more directly reflects the intended semantics of the query.

4.1 FISQL Examples

A full EBNF for FISQL appears in Figure 9. For brevity, we introduce FISQL by means of representative examples showing key transformations in the

[4]This becomes clear in the presentation of queries Q7 and Q8 in Lakshmanan et al. [2001].

```
⟨query⟩            ::=  SELECT ⟨col_decls⟩ INTO ⟨name_term⟩
                        FROM ⟨variable_decls⟩
                        [WHERE {⟨condition⟩}*]
                     |  (⟨query⟩) UNION (⟨query⟩)
                     |  (⟨query⟩) MINUS (⟨query⟩)
⟨col_decls⟩        ::=  ⟨col_decl⟩ {, ⟨col_decl⟩}*
⟨col_decl⟩         ::=  ⟨name_term⟩ AS ⟨string⟩
                     |  ⟨name_term⟩ ON ⟨name_term⟩
                     |  * [DROP ⟨name_term⟩ {, ⟨name_term⟩}*]
⟨variable_decls⟩   ::=  ⟨var_decl⟩ {, ⟨var_decl⟩}*
⟨var_decl⟩         ::=  ⟨db_name⟩ ⟨base_var_decl⟩
                     |  (⟨query⟩) ⟨base_var_decl⟩
⟨base_var_decl⟩    ::=  :⟨varname(rel)⟩:⟨varname(att)⟩ AS ⟨varname(tup)⟩
⟨condition⟩        ::=  (⟨condition⟩)
                     |  (⟨condition⟩) AND (⟨condition⟩)
                     |  (⟨condition⟩) OR (⟨condition⟩)
                     |  NOT (⟨condition⟩)
                     |  ⟨name_term⟩ ⟨cond_operator⟩ ⟨name_term⟩
⟨cond_operator⟩    ::=  = | != | <= | < | > | >=
⟨name_term⟩        ::=  ⟨string⟩
                     |  ⟨varname(meta)⟩
                     |  ⟨varname(tup)⟩.⟨varname(meta)⟩
                     |  ⟨varname(tup)⟩.⟨dom_elt⟩
                     |  ⟨varname(tup)⟩.⟨varname(tup)⟩.⟨dom_elt⟩
⟨varname(meta)⟩    ::=  ⟨varname(rel)⟩|⟨varname(att)⟩
⟨varname(X)⟩       ::=  ⟨dom_elt⟩ — X is rel, att, or tup
⟨db_name⟩          ::=  ⟨dom_elt⟩
⟨string⟩           ::=  "⟨dom_elt⟩"
⟨dom_elt⟩          ::=  (a–z|A–Z|0–9){(a–z|A–Z|0–9|-)}*
```

Fig. 9. EBNF grammar for Federated Interoperable SQL.

```
SELECT I.Dept AS "Dept", I.Store AS "Store"
      INTO "CompareChiIndy"
FROM Chicago.AvgSales AS C, Indy.Sales AS I
WHERE C.I.Dept > I.AvgSales
```

Fig. 10. Q_1: Use of tuple dereferencing.

`AmartSales` federation (Figure 1). The first example shows a transformation from the Chicago to Indianapolis representations.

Example 4.1. Consider query Q_1 (Figure 10) where we dereference the tuples in the input relation `Chicago.AvgSales` based on Indianapolis departments. This query returns departments in Indianapolis stores that are underperforming with respect to some Chicago counterpart. The result is a database having a single relation named `CompareChiIndy`.[5]

[5]Compare this schema-independent FISQL query to the schema-dependent SQL version in Figure 2.

```
SELECT I.Dept AS "Dept", I.AvgSal AS "AvgSal"
      INTO I.Store
FROM  Indy.Sales AS I
```

Fig. 11. Q_2: Indy.Sales $\mapsto$ Milw.

```
SELECT R AS "Store", T.Dept AS "Dept", T.AvgSales AS "AvgSales"
      INTO "Milw2Indy"
FROM Milw:R AS T
```

Fig. 12. Q_3: Milw $\mapsto$ Indy.Sales.

Two tuple variables are declared (C and I) over the Chi.AvgSales and Indy.Sales relations, respectively. The Dept value of the I tuple then gives the name of the *attribute* whose value in the C tuple is to be compared to I.AvgSales; in other words, the *dereferenced* value C.I.Dept is to be compared to I.AvgSales. In case I.Dept $\notin$ *schema*(Chi.AvgSales), the term C.I.Dept resolves to $\perp$.

As the previous example illustrates, in FISQL the terms "$T_1.T_2$.a" where T_1 and T_2 are tuple variables or "$T.M$" where T is a tuple variable and M is a metavariable capture the FIRA dereference operator, Δ. The next examples (4.2 through 4.4) show FISQL counterparts for the $\wp$, Σ, and τ operators, respectively.

Example 4.2. Query Q_2 (Figure 11) restructures the Indy.Sales relation into the format of the Milwaukee database.[6] A separate relation for each value of $\pi_{\texttt{Store}}$(Indy.Sales) is created within the output database due to the parameterized term "I.Store" occurring in the INTO part of the SELECT clause. This effects the same transformation as the FIRA $\wp$ operator depicted in Figure 7.

Example 4.3. Query Q_3 (Figure 12) exhibits the opposite transformation from query Q_2, effecting the same transformation as the FIRA generalized union operator, Σ.[7]

Example 4.4. Query Q_4 (Figure 13) restructures the Indianapolis data into the format of the Chicago data. For each value of $\pi_{\texttt{Sales}}$(Indy.Sales), a new column is created with appropriate contents. This is the same transposition depicted in Figure 8.

4.2 FISQL Semantics

In defining the semantics of FISQL, we assume an underlying *Federation Metadata Dictionary* (denoted FMD) which is a two-column relation containing the relation, and attribute names for each relation in the federation.[8] We will assume

[6]Since the Indy relation does not include store codes, these columns are omitted. They could be included containing null values for exact schema conversion.
[7]In an implementation, we would want to provide basic string manipulation capabilities in FISQL so that (for example) the string "LincAveSales" can be transformed to the string "LincAve", giving a more precise result.
[8]Commercial database platforms commonly store the required metadata in system specific tables; FMD can be easily obtained by querying these tables for each component database in the federation as long as appropriate privileges are held.

```
SELECT I.Store AS "Store", I.AvgSales ON I.Dept
      INTO "Indy2Chi"
FROM Indy.Sales AS I
```

Fig. 13. Q_4: `Indy.Sales` ↦ `Chi.AvgSales`.

```
SELECT X1 AS "a1", ..., Xk AS "ak", Y1 ON Z1, ..., Yp ON Zp
    INTO N
FROM D1 : R1 : A1 AS T1, ..., Dn : Rn : An AS Tn
WHERE C1 AND ··· AND Cm
```

Fig. 14. Canonical form of FISQL query.

```
SELECT I.Dept AS "Dept", I.Store AS "Store"
   INTO "CompareChiIndy"
FROM Chicago:R1:A1 AS C, Indy:R2:A2 AS I
WHERE R1 = "AvgSales" AND R2 = "Sales" AND C.I.Dept > I.AvgSales
```

Fig. 15. Query Q_1 from Figure 10 written longhand.

the schema of this relation is `FMD(relName,attName)`. `FMD` provides instantiations for FISQL metavariables at runtime.

A well-formed FISQL query has the form shown in Figure 14. In Figure 14, N, X_i, Y_i, and Z_i denote FISQL name terms conforming to the EBNF in Figure 9. The $\mathtt{a}_i$ are constants of ***dom***. Furthermore, $\mathtt{D}_i \in \mathfrak{M}_0$ are database names, and R_i, A_i, and T_i are relation metavariables, attribute metavariables, and tuple variables, respectively.[9]

Note that in the full form of a FISQL query, range declarations contain all metavariable declarations as well as tuple declarations. In Examples 4.1 through 4.4, we have made use of two convenient notational shortcuts in expressing `FROM` clauses, which we now elucidate.

First, variables that are not elsewhere used in the `SELECT` or `WHERE` clauses are normally omitted from the query. Second, it is often the case that specific relations are intended. In such a case, fixed relations are used directly within the `FROM` clause. The formal semantics involves unwinding these uses by replacing the relation name with a metavariable and adding an appropriate clause to the `WHERE` clause which constrains the metavariable. As an example, Figure 15 shows the long version of query Q_1. This long version can be given a precise, uniform semantics.

The semantics of a canonical FISQL query is given by a nested loop algorithm that instantiates each `FROM` clause metavariable using `FMD` and instantiates each tuple variable from the relation currently indicated by the metavariable instantiation. The algorithm is shown in Figure 16. Note that FISQL `FROM` clause declarations are independent, as in SQL (but unlike in SchemaSQL).

4.2.1 *Subqueries.* FISQL allows subqueries in the `FROM` clause, so we must extend the semantics of nested range declarations inductively to subqueries. We

[9]We have simplified the form somewhat by (i) assuming the `WHERE` clause is in Conjunctive Normal Form (CNF), (ii) omitting separate `SELECT` clauses for `*`, and (iii) putting `ON` terms after `AS` terms in the `SELECT` clause. These simplifications are for ease of presentation only and do not significantly impact either the semantics or the proof of equivalence with FIRA (Section 4.4).

for $i_1 \in$ FMD, **let** $R_1 = i_1$[relName], $A_1 = i_1$[attName]
...
for $i_n \in$ FMD, **let** $R_n = i_n$[relName], $A_n = i_n$[attName]
for v_1 in relation R_1
...
for v_n in relation R_n
if the WHERE clause is satisfied when instantiated values from v_1 through v_n and i_1 through i_n are substituted,
then evaluate the terms in the SELECT clause according to the instantiation and produce the tuple of values that results into the resulting relation named.

Fig. 16. Nested loops algorithm for evaluating FISQL queries.

begin at the innermost nested queries; by induction these return well-formed databases. Now consider a range declaration over a subquery, for example of the form: FROM (Q):R:A AS T. By induction, the subquery Q returns a database $\mathcal{D}_Q$. We obtain valid interpretations for this declaration by replacing FMD in the nested loops algorithm by the appropriate references from $\mathcal{D}_Q$.

Given this technique, we determine the range of each (meta) variable in the FROM clause of an arbitrary FISQL query, working from innermost to outermost queries. A similar nested loops algorithm to that in Figure 16 is used to instantiate the (meta) variables. The WHERE and SELECT clause semantics will not change.

4.3 Complexity of FISQL

PROPOSITION 4.5. *The data complexity of a fixed FISQL query Q is* LOGSPACE. *That is, given input federation $\mathcal{I}$, we can compute the result $Q(\mathcal{I})$ using $O(\log(|\mathcal{I}|))$ workspace in addition to that required to store the input and output.*

PROOF. Given a flat FISQL query that has n FROM clause declarations, consider how the nested loops computation of $Q(\mathcal{I})$ works. We need only maintain $3n$ pointers to the input federation (one for each tuple- and meta-variable in the FROM clause). Pointers corresponding to metavariables reference tuples in the Federation Metadata Dictionary (FMD) which is part of $\mathcal{I}$; pointers corresponding to tuple variables reference data tuples of $\mathcal{I}$. Each "pointer" is an index value numerically indicating a place in the input (a cell on an input tape for a Turing Machine, for example). It is well known that such numbers require logarithmic space to store (in terms of their magnitude).

Thus, for fixed Q, we require at most $3n$ pointers, independently of how many relations occur in the input federation or how large each is. This argument shows that the evaluation of a flat FISQL query is LOGSPACE. To see that the evaluation of an arbitrarily nested FISQL query is also LOGSPACE, consider the "innermost-to-outermost" evaluation of nested FISQL queries and note that the composition of LOGSPACE functions is LOGSPACE. □

This result is not particularly surprising, but it indicates that FISQL is a tractable language, as is canonical RA. The query complexity of FISQL will be significantly higher (as for SQL), however note that FISQL queries will in

general be short, and for many common tasks will be significantly shorter than SQL counterparts, as the queries in Figures 2 and 10 exemplify.

4.4 Equivalence of FIRA and FISQL

THEOREM 4.6

(1) *For every FISQL query Q there is an equivalent FIRA query $\hat{Q}$ such that for well-formed federation instances $\mathcal{F}$, $Q(\mathcal{F}) = \hat{Q}(\mathcal{F})$.*

(2) *For every FIRA query $\hat{Q}$ there is an equivalent FISQL query Q such that for well-formed federation instances $\mathcal{F}$, $\hat{Q}(\mathcal{F}) = Q(\mathcal{F})$.*

PROOF. From the presentation above, it should be reasonably clear that FISQL can express all FIRA queries. The interesting direction is the reverse, namely that FIRA can express all FISQL queries. In particular, this gives a formal, set-based (implementation independent) definition of the FISQL semantics.

To fix a notation for the translation, assume the input FISQL query is as in Figure 14. The translation from FISQL to FIRA parallels the SQL to RA translation and comprises five steps, as follows:

(1) Each `FROM` clause declaration has the form $\mathtt{D}_i : R_i : A_i$ `AS` T_i. (Note that we use the index i to indicate the position of the declaration in the `FROM` clause.) Given this, we translate the range declaration to the FIRA term $\Sigma(\downarrow_i (\mathtt{D}_i))$. As in the SQL to RA translation, the entire `FROM` clause translates to a Cartesian product of range terms, so that the result from this step has the form:
$$F := \Sigma(\downarrow_1 (\mathtt{D}_1)) \hat{\times} \cdots \hat{\times} \Sigma(\downarrow_n (\mathtt{D}_n)).$$
(Note that renaming may be required during this step as well.)

(2) The `WHERE` clause translates to a Boolean selection condition, so that this step results in a query of the form $S := \sigma_B(F)$. The condition B depends on the `WHERE` clause; the overline notation is particularly useful for a direct translation, so that Δ terms are absorbed into the selection (see the examples below).

(3) To translate the `SELECT` clause, we first project S on all columns appearing in the `SELECT` clause (including those arising within the `ON` construct on either side).

(4) Next, we perform the transpositions listed as `ON` terms in the `SELECT` clause. It can be shown that the order of the transpositions does not matter as long as the input query satisfies a certain well-definedness condition.[10]

(5) Finally, we drop the columns appearing in the `ON` constructs and partition to obtain $\hat{Q}$.

We illustrate the translation process on queries Q_1 (Figure 10) and Q_4 (Figure 13). □

[10]The condition is given pairwisely for transpositions as follows. Let $\mathcal{R}$ be a federated relation and let $A_1, A_2, B_1, B_2 \in \boldsymbol{dom} \cup \mathfrak{M}_1$. If $\pi_{B_1}(body(\mathcal{R}))$, $\pi_{B_2}(body(\mathcal{R}))$, and $schema(\mathcal{R})$ are pairwise disjoint, then $\tau_{A_1}^{B_1}(\tau_{A_2}^{B_2}(\mathcal{R})) = \tau_{A_2}^{B_2}(\tau_{A_1}^{B_1}(\mathcal{R}))$. If this condition does not hold, the $X \neq t[B]$ clause of Definition 3.17(1) can be used to enforce a desired order of transpositions.

Example 4.7. Recall query Q_1 (Figure 10) that finds the Indianapolis departments that make less than their Chicago counterparts. Query Q_1 translates to FIRA as follows:

(1) To resolve conflict between the `Indy.Sales` and `Chicago.AvgSales` relations, we rename the `Sales` attribute appropriately.[11] The `FROM` clause then translates as:

$$F \quad := \quad \Sigma(\downarrow_1(\rho_{\texttt{Sales}\to\texttt{CSales}}(\texttt{Chicago}))) \;\times\; \Sigma(\downarrow_2(\rho_{\texttt{Sales}\to\texttt{ISales}}(\texttt{Indy}))).$$

(2) Translating the `WHERE` clause results in:

$$S := \sigma_{\mathfrak{r}_1=\text{"}\texttt{AvgSales}\text{"} \wedge \mathfrak{r}_2=\text{"}\texttt{Sales}\text{"} \wedge \overline{\texttt{Dept}}=\texttt{AvgSales}}(F).$$

(3) The projection results in $P := \rho_{\texttt{CStore}\to\texttt{Store}}(\pi_{\texttt{Dept,CStore}}(S))$.
(4) Query Q_1 has no transpositions.
(5) Since there are no transpositions, there are no columns to drop, so we directly partition to obtain $\hat{Q}_1 := \wp_{\text{"}\texttt{CompareChiIndy}\text{"}}(P)$.

Example 4.8. Recall query Q_4 (Figure 13) restructuring the Indianapolis data into the format of the Chicago relation. This FISQL query translates to a FIRA query as follows.

(1) The `FROM` clause translates to $F := \Sigma(\downarrow_1(\texttt{Indy}))$.
(2) The `WHERE` clause translation results in $S := \sigma_{\mathfrak{r}_1=\text{"}\texttt{Sales}\text{"}}(F)$.
(3) The projection results in $P := \pi_{\texttt{Store,AvgSales,Dept}}(S)$.[12]
(4) There is only one transposition, which translates as $T := \tau^{\texttt{Dept}}_{\texttt{AvgSales}}(P)$. Note that this relation is exactly that depicted in Figure 8.
(5) Finally, we drop the `AvgSales` and `Dept` columns and partition to obtain the result: $\hat{Q}_4 = \wp_{\text{"}\texttt{Indy2Chi}\text{"}}(\mu_{\texttt{AvgSales,Dept}}(T))$.

COROLLARY 4.9. *FIRA operators can be implemented using space that is logarithmic in the size of the input federation and query.*

COROLLARY 4.10. *FISQL contains a sub-language equivalent to core SQL.*

By "core SQL", what is meant is the part of SQL that is equivalent to RA. Corollary 4.10 formally characterizes how FISQL extends SQL, in that the equivalent sub-language of FISQL has the same semantics as this core. Furthermore, Corollary 4.10 sets the stage for adding nonclassical SQL features such as aggregation to FISQL (see future work in Section 7).

5. QUERY PROCESSING IN FISQL/FIRA

The *query lifecycle* of a FISQL query is shown in Figure 17(a). For reference, the SQL query lifecycle is shown in Figure 17(b). This figure highlights a major

[11]In the presentation we use "understandable" names whereas a formal mechanism would use system-specific, uniquely generated symbols for this.

[12]Note that this results in a relation whose contents are identical to the original `Indy.Sales` relation. In the next section, we provide optimization rules to forgo the previous two steps when actually processing such a query.

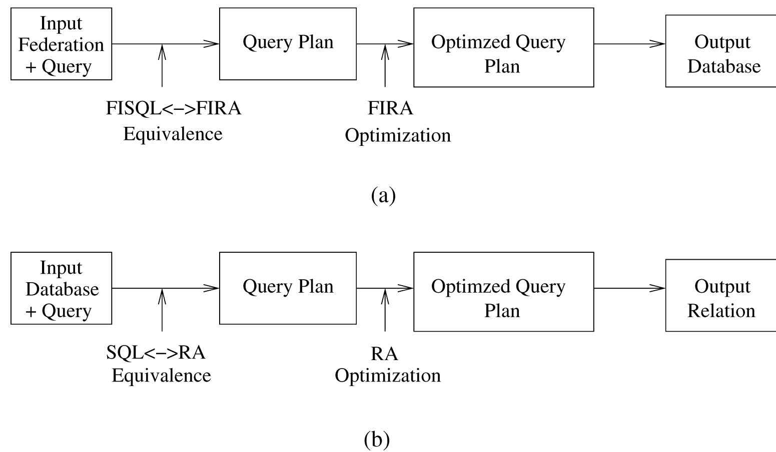

Fig. 17. Parallel query lifecycles.

strength of our framework, namely that the FISQL/FIRA lifecycle directly parallels the SQL/RA lifecycle. This direct parallel may seem "obvious" for FIRA and FISQL, but has been lacking in previous frameworks. In particular, the fact that RA is a subalgebra of FIRA allows us to utilize known query optimization techniques for the relational core of FISQL. General FISQL queries will no doubt require nonclassical extensions of these techniques. Future work will report on these extensions in the context of a FISQL prototype.

Figure 17 indicates that a *query plan* is the output of the FISQL/FIRA equivalence. Given an input FISQL query, Q, the query plan is essentially the parse tree for the equivalent FIRA query, $\hat{Q}$. In practice, this parse tree may be augmented with statistics concerning the data in the federation to assist in the optimization.

Given a query plan, optimization proceeds by transforming the FIRA expression tree according to algebraic equivalences with the goal of obtaining an optimally efficient query plan. In general, of course, this is a combinatorially difficult problem; however, several heuristic methods have been developed for RA expressions that work very well in practice—often decreasing the run time of a query by several orders of magnitude or more. We can identify two salient heuristic principles in the optimization of RA expressions which we can port into our FIRA framework without much difficulty (adapted from Ramakrishnan and Gehrke [2003]):

(1) We want to "push" selections (σ) and projections (π, μ) as far down the parse tree as possible in order to maximize their constraining effect.
(2) We want to combine selections and Cartesian products into *joins*, since joins are (in general) much more efficient than Cartesian products.

Note that since RA is isomorphic to a subalgebra of the FIRA, the algebraic identities used in RA optimization will have natural FIRA counterparts. These identities include such transformations as pushing selections (and projections) over unions and Cartesian products, or commuting Cartesian products. We will

assume these results without explicit statement (see Garcia-Molina et al. [2000] and Ramakrishnan and Gehrke [2003]). In addition (in line with heuristic principle (1) above), we will use the following identities which cover the cases of the new FIRA operators with respect to selection and projection.

PROPOSITION 5.1

(1) *Let $\mathcal{D}$ be a database and A, B, and C be distinct elements of* $\boldsymbol{dom} \cup \mathfrak{M}_1$*. Then*
   (a) $\sigma_{A=B}(\Sigma(\mathcal{D})) = \Sigma(\sigma_{A=B}(\mathcal{D}))$ *and*
   (b) $\sigma_{A=B}(\wp_C(\mathcal{D})) = \wp_C(\sigma_{A=B}(\mathcal{D}))$.
(2) *Let $\mathcal{D}$ be a database and A, B, C, and X be distinct elements of* $\boldsymbol{dom} \cup \mathfrak{M}_1$ *such that* $A, B \notin \pi_X(body(\mathcal{R}))$ *for any* $\mathcal{R} \in \mathcal{D}$*. Then*
$$\sigma_{A=B}\left(\Delta_X^C(\mathcal{D})\right) = \Delta_X^C(\sigma_{A=B}(\mathcal{D})).$$
(3) *Let $\mathcal{D}$ be a database and* $A_1, \ldots, A_n \in \boldsymbol{dom} \cup \mathfrak{M}_1$*. Then*
$$\pi_{A_1,\ldots,A_n}(\Sigma(\mathcal{D})) = \Sigma(\pi_{A_1,\ldots,A_n}((\mathcal{D}))).$$
(4) *Let $\mathcal{D}$ be a database and* $A_1, \ldots, A_n \in \boldsymbol{dom} \cup \mathfrak{M}_1$*. Let* $X \in \boldsymbol{dom} \cup \mathfrak{M}_1$ *be distinct from each* A_i $(1 \leq i \leq n)$*. Then*
$$\pi_{A_1,\ldots,A_n,X}(\wp_X(\mathcal{D})) = \wp_X(\pi_{A_1,\ldots,A_n,X}(\mathcal{D})).$$
(5) *Let $\mathcal{D}$ be a database. Let* $A_1, \ldots, A_n, C, X \in \boldsymbol{dom} \cup \mathfrak{M}_1$ *be distinct elements such that* $A_i \notin \pi_X(\mathcal{R})$ *for* $1 \leq i \leq n$ *and* $\mathcal{R} \in \mathcal{D}$*. Then*
$$\pi_{A_1,\ldots,A_n,X}\left(\Delta_X^C(\mathcal{D})\right) = \Delta_X^C(\pi_{A_1,\ldots,A_n,X}(\mathcal{F})).$$

The proof of these equivalences is not difficult, and is omitted for brevity.

So far, FISQL optimization parallels SQL optimization. However, there is at least one important optimization paradigm that this does not capture. Note that we have defined the generalized union operator (Σ) as essentially unioning *entire* input databases, whereas most queries will only mention a subset of these relations. Given a database $\mathcal{D}$ and $N \in \mathfrak{M}_0$, let the notation $\mathcal{D}|_N$ represent the *restriction of $\mathcal{D}$ to the relation named N* (*i.e.* $\{\mathcal{R}\}$ where $\mathcal{R} \in \mathcal{D}$ and $name(\mathcal{R}) = N$). In this case, we have the following additional rewrite rule:

LEMMA 5.2. *Let $\mathcal{D}$ be a database and* $X \in \mathfrak{M}_0$*. Then* $\sigma_{\mathfrak{r}_i = "X"}(\Sigma(\downarrow_i (\mathcal{D}))) = \downarrow_i (\mathcal{D}|_X)$.

This rule enables us to restrict the input federation on the basis of the relations that are actually used in the query.

As an illustration, we provide a detailed example of how to use these identities when optimizing a FISQL query plan.

Example 5.3. Consider the query shown in Figure 18. This query finds those departments whose average monthly sales exceeds \$10,000 for at least one store in every region. This FISQL query translates to the (unoptimized) FIRA query, $\hat{Q}$ (Figure 19). The parse tree for $\hat{Q}$ is shown in Figure 20(a).

To optimize the initial FIRA query, we can push selections down the tree according to the algebraic identities from Proposition 5.1 and Ramakrishnan and Gehrke [2003]. We can also combine Cartesian products with selections to form joins. The resulting optimized FIRA query plan is shown in Figure 20(b). In

```
SELECT T1.Dept INTO result
FROM Indy:R1:A1 AS T1, Chi:R2:A2 AS T2, Milw:R3:A3 AS T3
WHERE R1 = "Sales" AND R2 = "AvgSales" AND T1.AvgSales > 10000 AND
    T2.T1.Dept > 10000 AND T3.Dept = T1.Dept AND T3.AvgSales > 10000
```

Fig. 18. FISQL query Q to be optimized.

$$\wp_{\texttt{"resRel"}}\left(\pi_{\texttt{T}_1\texttt{.Dept}}\left(\sigma_{\substack{\mathfrak{r}_1=\texttt{"Sales"} \wedge \mathfrak{r}_2=\texttt{"AvgSales"} \wedge \\ \texttt{T}_1\texttt{.AvgSales} > 10000 \wedge \overline{\texttt{T}_1\texttt{.Dept}} > 10000 \wedge \\ \texttt{T}_3\texttt{.Dept}=\texttt{T}_1\texttt{.Dept} \wedge \texttt{T}_3\texttt{.AvgSales} > 10000}} \left(\Sigma(\downarrow_1 (\texttt{Indy})) \times \Sigma(\downarrow_2 (\texttt{Chi})) \times \Sigma(\downarrow_3 (\texttt{Milw}))\right)\right)\right)$$

Fig. 19. Unoptimized algebraic query $\hat{Q}$ equivalent to Q.

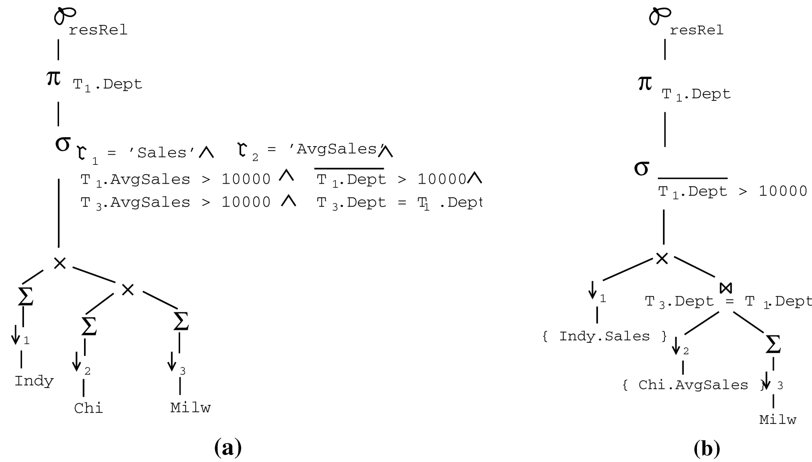

Fig. 20. Query plan for Q translated into FIRA (a) and optimized (b).

practice, the execution plan shown in Figure 20(b) is likely to be several orders of magnitude faster than that shown in Figure 20(a). The ease with which the SQL/RA optimizations are formally ported to the FISQL/FIRA framework is one of our main contributions to the field.

Note that in the case of the Indianapolis and Chicago databases, the selections appearing after Σ operations in fact reduce the input to single-relation databases. However, in the Milwaukee database, we retain all input relations.[13]

Additionally, we could eagerly project off relation columns that will not be used in the query plan (this is standard practice, but not shown in Figure 20(b)).

Note that, in practice, statistics of the data and information about existing indices would allow us to further optimize the query, for example by rewriting the join ordering optimally. However, this example illustrates that FIRA provides the ability to optimize FISQL queries utilizing existing SQL/RA methods.

[13]In an implementation, we could provide a LIKE operator, and extract relations from Milwaukee having a name LIKE "Sales".

Implementing the relational operators is well understood. Highly efficient algorithms exist for $\sigma, \pi, \cup$ and $-$. In practice, $\times$ is usually implemented as a *join* ($\bowtie$), which is a combined selection and product. This enables us to achieve performance significantly better than the quadratic time suggested by the behavior of a product. We will not pursue the relational algorithms here; there are a number of good resources for this [Garcia-Molina et al. 2000; Ramakrishnan and Gehrke 2003]. Furthermore, the new FIRA operators μ, Σ, $\wp$ and Δ can clearly utilize the sorting and/or hashing techniques employed by the relational algorithms, obtaining linear (or better) performance. Ongoing work is discovering innovative data structures and algorithms (or reframing existing structures and algorithms) for use in the FISQL/FIRA framework. As an example, distributed query processing methods commonly utilize an analog of Lemma 5.2.

6. RELATED WORK

In this section, we discuss previous work that is related to our query languages. Our emphasis is on languages that support relational metadata integration. As such, we consider only briefly representative approaches based on complex object or semistructured data models. Most recent wrapper generation and/or mediator specification languages fall into the latter categories [Lenzerini 2002].

Within the area of relational metadata integration languages, we use several criteria for comparison (see Section 6.14 for a summary of these). Foremost among these is the notion of transformational completeness (Section 2.2). This notion breaks down into two distinct concepts: *schema independence* and *dynamic schemas*. Most languages we consider below allow schema independent queries to some degree, where the user can formulate queries without perfect knowledge of the input schema. A consequence of this is that the language supports at least some types of schema evolution gracefully.

Beyond schema independence, the FIRA τ and $\wp$ operators are capable of producing fully dynamic output schemas, where the number of relations and/or the number of columns in relations of the output varies dynamically with the input instance. Of the languages surveyed, only FISQL/FIRA provides generalized dynamic output schemas, where the values in any (fixed) number of input columns can determine output schemas.

Other criteria for comparison include the lowest published complexity of the language, whether the language supports nested queries, and whether the language has an equivalent (published) algebra. For a tabular summary of the languages surveyed see Section 6.14.

6.1 HiLog

HiLog appeared as a database programming alternative in the late 1980s [Chen et al. 1989, 1990b]. The premise was to extend ordinary first order predicate logic (which underlies SQL and the RA) with second order capabilities so that metadata could be included in queries seamlessly with data. HiLog had its roots in F-logic [Kifer and Lausen 1989] and improved on its contemporaries (COL [Abiteboul and Grumbach 1990] and LDL [Naqvi and Tsur 1989]) in that it provided second order query functionality within a first order semantics.

This essentially meant that HiLog eliminated some of the unwieldy constraints arising from second-order well-foundedness criteria. Furthermore, the idea of a second-order syntax with a first-order semantics emerged as a foremost candidate for interoperable query languages as a result of HiLog [Chen et al. 1990a].

Although HiLog was a significant step forward, it has some limitations from our point of view. HiLog was developed as a platform for supporting object-oriented database languages, rather than purely relational ones. Also, HiLog syntax is logical rather than declarative (it is based on Prolog). HiLog allows recursion, which means it is inefficient according to current standards for database query languages. HiLog has no equivalent algebra and no associated query optimization framework.

Finally, a more serious limitation is that terms in HiLog have fixed arity so that a query returning (for example) all relations (of all arities) that mention the atom `'Indy'` cannot be phrased in HiLog.

6.2 Multirelational Algebra

A precedent for some of our federated relational operators arose in work by Grant, Litwin, Roussopolous and Sellis in the late 1980s and early 1990s on MSQL Litwin et al. [1989] and a corresponding algebra for relational multidatabases [Grant et al. 1993], which we term here the *Multirelational Algebra*, or MRA. MSQL is a version of SQL developed for multidatabases. In MSQL, any number of relations can be queried. MSQL allows for varying relation arities and thus supports schema independent querying. MSQL includes facilities for aggregation, view creation, and dynamic databases. However, MSQL does not permit the restructuring of data into metadata on the fly, so that, for example, the transformation of the `Indy.Sales` relation into the format of `Chi.AvgSales` is not possible in MSQL.

MSQL queries can be optimized using the MRA. The MRA contains extensions of the relational operators Π, σ, $\bowtie$, $\cup$, and $\cap$ for multirelations. These operators are termed MPROJECT, MSELECT, MJOIN, MUNION, and MINTERSECT, respectively. The operators MPROJECT and MSELECT are essentially identical to our federated operators $\hat{\Pi}$ and $\hat{\sigma}$. However, MJOIN, MUNION, and MINTERSECT results are defined to contain all pairwise combinations possible in the input multirelations. This is possible in the MRA because relations may be *unnamed*. One major consequence of this decision is that several desirable properties of the RA do not carry over to the MRA. For example, MJOIN need not be associative, and MPROJECTs need not cascade [Grant et al. 1993].

In contrast, these desirable RA properties do carry over to FIRA. The decision to use federated versions of $\bowtie$, $\cup$, and $\cap$ that only modify like-named relations is natural in the FISQL/FIRA framework, and entails that the behavior of our operators remains closer to that of their original relational counterparts.

6.3 Interoperable Database Language

The Interoperable Database Language (IDL) is a Horn clause based language for interoperability and schema integration developed shortly after HiLog [Krishnamurthy et al. 1991]. IDL extends the Horn clause language developed

by Krishnamurthy and Naqvi [1998], which in turn extends LDL [Naqvi and Tsur 1989]. Thus, IDL may be seen as the result of a series of attempts to refine the LDL framework; in this effort, many of the concerns posed within the development of HiLog were addressed. IDL also has roots within the language MSQL.

As with many of the frameworks for schema independent querying developed in the 1990s, IDL relies on a complex object model wherein metadata are explicitly represented within a nested object structure. IDL also allows data objects to be *updated*.

IDL does not allow recursion, so its complexity will fall below that of HiLog. On the other hand, IDL allows negation in query terms, so the issue of safety arises.

In IDL, output relations have fixed arity. Thus, in Krishnamurthy et al. [1991], interoperability within a federation such as `AmartSales` is accomplished by providing normalized views transforming data in formats such as `Milw` and `Chicago` into a format much like that of `Indy.Sales`. (The reverse transformations are not provided.)

Additionally, the fact that IDL has a logic-based syntax and object based semantics is problematic for integrating the IDL framework with existing RDBMSs.

6.4 The Ross Algebra

Ross [1992] extended the relational algebra with two families of operators capable of metadata querying. These families are the *totality* operators, ρ_k, that return all names of relations with arity k; and the *expansion* operators, $\alpha^k(R)$, which return the union of the k-ary relations named in R, coupled with their names. Thus, $\alpha^k(R)$, is a restricted form of Σ and $\downarrow$, where the input is a single column of relation names and the output is restricted to arity $k+1$. The main reason for the arity restriction is that Ross desires the relations be *union compatible* in the RA sense (so their schemas must match); in the federated model, we have extended the notion of union compatible to arbitrary relations.

In the Ross algebra, relation names may be demoted and manipulated as data. Relation names are simple domain elements. Ross provides an equivalent calculus for the algebra, and shows that his algebra can be expressed in the standard relational algebra in case the database contains "universal relations" explicitly coupling the relation names with contents (these are essentially the results of our $\downarrow$ operator applied to the database). The Ross algebra thus provides an elegant framework for querying relation names; however, the framework is limited to the demotion of atomic relation names, and does not support the other transformations of Figure 3.

6.5 SchemaLog

SchemaLog [Lakshmanan et al. 1997, 1993] is another logic programming language supporting metadata querying and interoperability, in the vein of LDL and HiLog. Like HiLog, SchemaLog has a second-order syntax but first-order semantics. Furthermore, SchemaLog has a well developed proof theory based

on the first order predicate calculus [Lakshmanan et al. 1997]. SchemaLog supports recursion and so is inefficient as a query language without restrictions.

A significant advance in SchemaLog is the removal of the fixed arity restriction of the Ross framework and HiLog. In SchemaLog, a single variable can unify with a list of objects, providing dynamic schema creation. This syntactic *untypedness* leads to a form of ambiguity where the difference between a dynamic output schema and static output schema cannot be determined within the head of a query clause. (In contrast, we provide special keywords "ON" and "INTO" for signaling dynamically varying schemas.)

In Lakshmanan et al. [1997], an extended relational algebra is developed for a fragment of SchemaLog. This algebra (known as the ERA) contains facilities for obtaining metadata from a federation, as well as a powerful operator, γ, for matching SchemaLog-type *patterns*. The ERA has an equivalent calculus, and is shown to be equivalent to the "querying fragment" of SchemaLog. Dereference and transposition operations cannot be performed within the ERA, so it is not transformationally complete in the sense of Figure 3. The ERA is extended in Andrews et al. [1996] to include procedural mapping operators which can transform relations according to "programming relations", an idea which seems to recall the reflective programming model (Section 6.7, below).

A simplified version of SchemaLog known as *WebLog*, appears in Lakshmanan et al. [1996a] for use in querying and restructuring data on the World Wide Web.

6.6 Generalization/Aggregation Relational Algebra

An interesting precursor to our work is the extended relational algebra obtained in Saltor et al. [1993] by adding operators that perform what the authors term *generalization* and *aggregation* functions. Upon inspection, the added operators are in fact restricted versions of FIRA operators. The generalization/aggregation operators are added particularly to introduce the data-metadata transformations in an example federation very similar to the AmartSales federation. The example produces the following operators: *(inner) discriminated union*, *partition by attribute*, *decomposition*, and *composition*. These are, in fact, restricted versions of the FIRA Σ, $\wp$, Δ, and τ operators, respectively. The *outer* version of the discriminated union is essentially the full Σ operation. It is interesting to note that problems with the shape of transposition results are overcome in Saltor et al. [1993] by restricting the input relation to a certain shape (in particular, only one "non-key" attribute).

We denote the extended algebra developed in Saltor et al. [1993] using the term *Generalization/Aggregation Relational Algebra*, or G/A RA for short. The G/A operators are extended in Saltor et al. [1993] to an object-oriented model, however restrictions remain. No associated declarative (SQL-like) query language is developed for the G/A framework.

6.7 Relational Languages Utilizing Reflection

Previous frameworks discussed involved augmenting the data model (usually with nested objects) and/or raising the order of the query language (at least

syntactically). A different approach involves adding *reflective* capabilities to SQL [Van den Bussche et al. 1993]. In this approach, programs (queries) can create, manipulate, and evaluate programs themselves. This idea led to the *Reflective Relational Algebra* (RRA) of Van den Bussche et al. [1993] and *Reflective SQL* (RSQL) of Dalkilic et al. [1996], as well as (more recently) SISQL [Masermann and Vossen 2000] and Meta-SQL [Van den Bussche et al. 2004].

Adding reflective capabilities to the relational model essentially allows queries to create *dynamic* queries, which take into account the state of the database at the time they are evaluated. As such, these languages can express all PTIME queries (more powerful versions exist, for example the *Recursive RRA* of Jain [1996]).

It is interesting to note that since the underlying query languages (RA or SQL) allow only fixed schema output, the reflective versions cannot produce truly dynamically varying schemas. This limitation is overcome by *simulating* dynamically varying output, for example by producing triples over attributes $\{rel, att, value\}$ where *rel* contains the relation name of a resulting tuple, *att* names the corresponding attribute, and *value* contains the corresponding data value. For example, using this schema, Masermann and Vossen [2000] contains a transformation that results in a relation that simulates the union of relations with incompatible schemas. Although these languages can semantically simulate dynamically varying schemas, they are not transformationally complete in our sense since relations whose attributes varies with the input data cannot be produced directly.

6.8 The Uniform Data Model

We have mentioned that frameworks for data-metadata integration may extend the data model using nested objects. At least one framework achieves relational metadata integration without recourse to nested structures: the *Uniform Data Model*, or UDM, of Jain [1996]. In UDM, there is no separation between data and metadata; essentially, every piece of metadata and data is compiled into binary *information relations*, which explicitly tag domain elements with their use in the database instance. Query languages for UDM include a powerful calculus and algebra; we can compare our work to the *safe* versions of these query languages (the Safe Uniform Calculus and Safe Extended Uniform Algebra, respectively). These languages can simulate transformationally complete querying within UDM, and the associated safe query languages are expressive yet feasible (in particular, LOGSPACE). However, in practice, it will be onerous to compile existing relational databases into UDM; furthermore, many queries that are naturally simple in the relational model become complex and unnatural in UDM. Nevertheless, UDM sets a notable precedent and is also reminiscent of the "simulation" approach toward dynamically varying schemas taken by the Reflective languages (Section 6.7).

6.9 SchemaSQL

SchemaSQL [Lakshmanan et al. 1996b, 2001] is a declarative language allowing schema-independent querying, and serves as an SQL-like counterpart for

a restricted subset of SchemaLog. SchemaSQL syntax is defined by example and provides transformations similar to FISQL, including a restricted version of τ where one column of data can be promoted to attribute values per query. SchemaSQL does not allow nested queries, and use of the `CREATE VIEW` construct in SchemaSQL changes the semantics of queries (making query compositionality problematic). SchemaSQL has no equivalent algebra, and unfortunately includes a MERGE operation in its semantics that results in nondeterministic behavior [Lakshmanan et al. 2001]. Recent attempts to pin down the semantics of SchemaSQL using a relational version of the tabular algebra (below) seem to rely on stored semantic knowledge [Lakshmanan et al. 1999]. Nevertheless, SchemaSQL is a conceptual ancestor of FISQL, so we have indicated throughout Section 4 where the design of FISQL diverges from that of SchemaSQL (and why).

6.10 The Tabular Algebra

The Tabular Algebra [Gyssens et al. 1996] formalizes operations on spreadsheet-like tables. The underlying data model is strictly richer than the relational model, but values in a table are atomic (as opposed to object-based). All FIRA operations can be imitated in the Tabular Algebra. The full Tabular Algebra includes a looping construct, and a formal result proves this full algebra can achieve *any* transformation between tabular data. We are currently investigating whether the nonlooping core of the Tabular Algebra can be imitated in FIRA (under a suitable mapping between the relational and tabular data models).

6.11 MD-SQL and MQL/MA

MD-SQL [Rood et al. 1999] shares most of the objectives of FISQL. MD-SQL is equivalent to a two-tiered meta-algebra containing a generalized join capable of joining an arbitrary number of input relations in a single query. This ability causes a leap in complexity to PSPACE. MD-SQL has limited subquery capabilities and, like SchemaSQL [Lakshmanan et al. 2001], includes "nested variable declarations," which must be unwound before an equivalent algebraic query can be constructed.

The language MQL [Wyss and Van Gucht 2001; Wyss et al. 2001] is an evolutionary step beyond MD-SQL, in that dynamic schemas can be created with the "ON" construct in the SELECT clause (as in FISQL), instead of the generalized join of MD-SQL. In the algebra equivalent to MQL, the Meta-Algebra (MA), the "ON" construct translates to the τ operator, as in FISQL/FIRA. However, MQL/MA is two-tiered, like MD-SQL, and contains an unweildy "$\mathfrak{Map}$" operation for producing dynamic output databases. This operator makes query optimization after the RA/SQL fashion difficult. The complexity of MQL is LOGSPACE (without the generalized join construct).

6.12 Complex-Object-Based Approaches to Metadata Integration

Many previous frameworks for data integration allow schema independent querying within a complex object data model. Examples include MetaOQL

[Su et al. 2000] and the TSIMMIS framework [Chawathe et al. 1994]. There are many more such systems, but the two mentioned seem relatively representative of the approach.

MetaOQL was developed to address the need to support complex schema transformations in OQL [Su et al. 2000]. In particular, the developers of MetaOQL were interested in supporting schema evolution in the SERF system [Claypool et al. 1998]. MetaOQL uses a syntax somewhat similar to SchemaSQL. MetaOQL queries can be translated at run time into ordinary OQL queries.

The TSIMMIS project is a collaboration between Stanford and the IBM Almaden Research Center to develop a framework for facilitating the rapid integration of heterogeneous information [Chawathe et al. 1994]. This project is much wider in scope than ours, and includes both structured and unstructured data from a wide range of sources. As such, there was a need within TSIMMIS to develop a framework for declaring metadata independent transformations between data sources. *Mediator Specification Language*, or MSL, arose to meet this need [Papakonstantinou et al. 1996]. MSL was designed to address problems of rapidly evolving schemas as well as schema discrepancy. The result is a powerful, rule-based language for specifying transformations between constituent data sources. Although some form of "algebraic" optimization is possible within the MSL framework, based on rule rewriting, there is no corresponding pure algebra (such as FIRA). Underpinning the MSL framework is a common data model based on complex objects known as the *Object Exchange Model*.

6.13 XML-Based Approaches to Metadata Integration

More recent frameworks for data integration are based on a semistructured (tree-like) model and involve use of XML-based technologies to achieve schema independent querying within that model. Examples of the approach include MIX [Baru et al. 1999] and Lixto [Gottlob et al. 2004]. Many more such systems for wrapper generation and data integration on the Web exist, but we restrict our attention to the two mentioned as representatives of the approach.

Both MIX and Lixto target data integration on the web, where queries are issued to form interfaces and responses are (X)HTML documents of varying schemas (often utilizing visual-based tables for layout purposes). Within this context, MIX provides mediator views of XML DTDs in an XML query language termed *XML Matching and Structuring Language* (XMAS). In contrast, Lixto utilizes a mediator language called *Elog* which is based on second-order monadic Datalog over trees [Gottlob et al. 2004]. Both MIX and Lixto focus on providing sophisticated graphical interfaces for web wrapper generation, since XML query languages can seem overly complicated to many users.

One of the first points which becomes obvious when dealing with XML data is that "metadata integration" needs to have a different meaning. Many XML query languages naturally allow one to compare data (such as text) with element tags (metadata). There are many fruitful avenues for work on XML-querying after the "FISQL style", such as characterizing what transformational

Table I. Comparison of Frameworks for Relational Metadata Integration

Framework	Properties					
	EA	SI	DOS	PX	Det	NQ
SQL/RA	√			LS	√	√
FISQL/FIRA	√	√	√	LS	√	√
HiLog		√		TC	√	√
MSQL/MRA	√	√			√	
Ross Alg/Calc	√	√		LS	√	√
SchemaLog		√	√	TC	√	√
G/A RA	√	√	√		√	
RSQL/RRA/SISQL	√	√		PT	√	√
SchemaSQL		√	√			
MD-SQL	√	√	√	PS	√	
MQL/MA	√	√	√	LS	√	

completeness involves for semi-structured data models. We defer a more detailed discussion to future work.

6.14 Summary of Related Work

In Table I, we provide a brief summary of related work. In the table, only query languages that have been used for the *relational* model are reflected. The summary is based on the following six criteria.

(1) *Equivalent Algebra* (*EA*)—Is there a (published) query algebra for the framework, in particular one that can be used in query optimization.
(2) *Schema Independence* (*SI*)—Can the framework support queries that are (at least somewhat) independent of the input schema.
(3) *Dynamic Output Schemas* (*DOS*)—Does the framework support the creation of a varying number of relations and/or relations with a varying number of attributes.
(4) *Published compleXity* (*PX*)—The lowest published complexity is listed here (LS = LOGSPACE, PS = PSPACE, PT = PTIME, TC = Turing complete).
(5) *Deterministic* (*Det*)—This property is often left unsaid, but it is important: does the framework output a deterministic, single result in response to a query on an (unordered) relational database.
(6) *Nested Queries* (*NQ*)—Finally, does the framework allow the composition of queries. This is important from a theoretical standpoint, as well as in terms of easily supporting source-level optimizations in a data integration framework.

7. CONCLUSION AND FUTURE WORK

In this article, we have presented a formal paradigm for relational metadata integration consisting of the query languages FIRA and FISQL. Since we have established a formal syntax, semantics, and equivalence for these languages, we can more easily investigate important theoretical properties of the paradigm and compare FISQL/FIRA to existing languages and frameworks.

In addition, we have contributed (i) a formal relational data model for supporting metadata integration, termed the federated data model; (ii) a formalization of the notion of "transformational completeness" along the lines of the notion of relational completeness; and (iii) an extended survey of previous languages for relational metadata integration.

We have indicated throughout the article directions for ongoing and future work on FISQL/FIRA. These include (i) extending the languages with aggregation, (ii) extending BP-Completeness and related symmetry notions to the federated model and FIRA, (iii) characterizing when "merging" is well-defined in federated relations, (iv) characterizing the extended "relational hierarchy" beyond federations, (v) adopting existing data structures and algorithms for query processing in FISQL/FIRA (or inventing new structures and algorithms), and (vi) extending the notion of transformational completeness to other data models such as semi-structured or complex objects.

In general, future work on FISQL/FIRA will involve both implementation and theory. In terms of implementation, we are investigating FISQL/FIRA query processing within larger data integration networks based on centralized and peer-to-peer topologies. In terms of theory, we are working on formally comparing FISQL/FIRA to other frameworks such as the Tabular Algebra. The Tabular Algebra is particularly appealing because of formal results that the algebra encompasses all possible tabular data transformations. It may be that a Tabular Algebra "core" is equivalent to FIRA; in which case, we can postulate a more refined notion of transformational completeness. Similar results characterizing the relationship of FIRA to the nested relational algebra are also being investigated.

ACKNOWLEDGMENTS

We thank the anonymous TODS reviewers for their detailed and invaluable comments on previous drafts. Thanks also go to both Dirk Van Gucht and Felix Wyss, who were instrumental in previous work that helped to lay the ground for FISQL/FIRA. Also, thanks go to the members of the Indiana University database lab for input on countless prior versions of this work.

REFERENCES

ABITEBOUL, S. AND GRUMBACH, S. 1990. COL: A logic-based language for complex objects. In *Advances in Database Programming Languages, Papers from DBPL-1, September 1987, Roscoff, France*, F. Bancilhon and P. Buneman, Eds. ACM Press/Addison-Wesley, New York, 347–374.

ABITEBOUL, S., HULL, R., AND VIANU, V. 1995. *Foundations of Databases*. Addison-Wesley, New York.

ANDREWS, A. J., SHIRI, N., LAKSHMANAN, L. V. S., AND SUBRAMANIAN, I. N. 1996. On implementing SchemaLog—A database programming language. In *Proceedings of the 5th International Conference on Information and Knowledge Management* (Baltimore, Md.). ACM, New York, 309–316.

BARU, C., GUPTA, A., LUDÄSCHER, B., MARCIANO, R., PAPAKONSTANTINOU, Y., VELIKHOV, P., AND CHU, V. 1999. XML-based information mediation with MIX. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data* (Philadelphia, Pa.). ACM, New York, 597–599.

CHAWATHE, S., GARCIA-MOLINA, H., HAMMER, J., IRELAND, K., PAPAKONSTANTINOU, Y., ULLMAN, J. D., AND WIDOM, J. 1994. The TSIMMIS project: Integration of heterogeneous information sources. In

Proceedings of the 16th Meeting of the Information Processing Society of Japan. IPSJ, Tokyo, Japan, 7–18.

CHEN, W., KIFER, M., AND WARREN, D. S. 1989. HiLog: A first-order semantics for higher-order logic programming constructs. In *Logic Programming, Proceedings of the North American Conference 1989*, E. L. Lusk and R. A. Overbeek, Eds. (Cleveland, Ohio). MIT Press, Cambridge, Mass., 1090–1114.

CHEN, W., KIFER, M., AND WARREN, D. S. 1990a. HiLog: A foundation for higher-order logic programming. Tech. Rep., State University of New York at Stony Brook, Stony Brook, N.Y.

CHEN, W., KIFER, M., AND WARREN, D. S. 1990b. HiLog as a platform for database languages. In *Proceedings of the 2nd International Workshop on Database Programming Languages*, R. Hull, R. Morrison, and D. W. Stemple, Eds. (Gleneden Beach, Ore.). Morgan-Kaufmann, San Francisco, Calif., 315–329.

CLAYPOOL, K., JIN, J., AND RUNDENSTEINER, E. A. 1998. SERF: Schema evolution through an extensible, re-usable and flexible framework. Tech. Rep. WPI-CS-TR-98-9, Worcester Polytechnic Institute.

CODD, E. 1970. A relational model of data for large shared data banks. *Commun. ACM 13*, 60, 377–387.

CUNNINGHAM, C., GALINDO-LEGARIA, C. A., AND GRAEFE, G. 2004. PIVOT and UNPIVOT: Optimization and execution strategies in an RDBMS. In *Proceedings of the 30th International Conference on Very Large DataBases* (*VLDB 2004*) (Toronto, Ont., Canada). Morgan-Kaufmann, San Francisco, Calif., 998–1009.

DALKILIC, M., JAIN, M., VAN GUCHT, D., AND MENDHEKAR, A. 1996. Design and implementation of reflective SQL. Tech. Rep. TR 451, Indiana University.

GARCIA-MOLINA, H., ULLMAN, J. D., AND WIDOM, J. 2000. *Database System Implementation*. Prentice-Hall, Upper Saddle River, New Jersey.

GINGRAS, F. AND LAKSHMANAN, L. V. 1998. nD-SQL: A multi-dimensional language for interoperability and OLAP. In *Proceedings of 24th International Conference on Very Large Data Bases*, A. Gupta, O. Shmueli, and J. Widom, Eds. (New York City, N.Y.). Morgan-Kaufmann, San Francisco, Calif., 134–145.

GOTTLOB, G., KOCH, C., BAUMGARTNER, R., HERZOG, M., AND FLESCA, S. 2004. The lixto data extraction project—Back and forth between theory and practice. In *Proceedings of the 23rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (*PODS 2004*) (Paris, France). ACM, New York, 1–12.

GRANT, J., LITWIN, W., ROUSSOPOULOS, N., AND SELLIS, T. 1993. Query languages for relational multidatabases. *VLDB J. 2*, 153–171.

GYSSENS, M., LAKSHMANAN, L. V., AND SUBRAMANIAN, I. N. 1996. Tables as a paradigm for querying and restructuring. In *Proceedings of the 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. (Montreal, Qur., Canada). ACM, New York, 93–103.

HALEVY, A. 2004. Structures, semantics, and statistics. Keynote Address. In *Proceedings of the 30th International Conference on Very Large Databases* (Toronto, Ont., Canada). 4–6.

JAIN, M. 1996. Database models and query languages for relational data and metadata query processing. Ph.D. dissertation, Indiana University.

KIFER, M. AND LAUSEN, G. 1989. F-Logic: A higher-order language for reasoning about objects, inheritance, and scheme. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, J. Clifford, B. G. Lindsay, and D. Maier, Eds. (Portland, Ore., May 31–June 2). ACM, New York, 134–146.

KRISHNAMURTHY, R., LITWIN, W., AND KENT, W. 1991. Language features for interoperability of databases with schematic discrepancies. In *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data* (Denver, Colo.). ACM, New York, 40–49.

KRISHNAMURTHY, R. AND NAQVI, S. A. 1988. Towards a real horn clause language. In *Proceedings of the 14th International Conference on Very Large Data Bases*, F. Bancilhon and D. J. DeWitt, Eds. (Los Angeles, Calif., Aug. 29–Sept. 1). Morgan-Kaufmann, San Francisco, Calif., 252–263.

LAKSHMANAN, L. V., SADRI, F., AND SUBRAMANIAN, I. N. 1993. On the logical foundations of schema integration and evolution in heterogeneous database systems. In *Proceedings of the 3rd International Conference on Deductive and Object-Oriented Databases*, S. Ceri, K. Tanaka, and S. Tsur, Eds. (Phoenix, Az.). Springer-Verlag, New York, 81–100.

LAKSHMANAN, L. V. S., SADRI, F., AND SUBRAMANIAN, I. N. 1996a. A declarative language for querying and restructuring the web. In *Proceedings of the International Workshop on Research Issues in Data Engineering* (New Orleans, La.). IEEE Computer Society Press, Los Alamitos, Calif., 12–19.

LAKSHMANAN, L. V. S., SADRI, F., AND SUBRAMANIAN, I. N. 1996b. SchemaSQL—A language for interoperability in relational multi-database systems. In *Proceedings of 22th International Conference on Very Large Data Bases*, T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, Eds. (Mumbai, India). Morgan-Kaufmann, San Francisco, Calif., 239–250.

LAKSHMANAN, L. V. S., SADRI, F., AND SUBRAMANIAN, I. N. 1997. Logic and algebraic languages for interoperability in multidatabase systems. *J. Logic Prog. 32*, 2 (Nov.), 101–149.

LAKSHMANAN, L. V. S., SADRI, F., AND SUBRAMANIAN, S. N. 1999. On efficiently implementing SchemaSQL on a SQL database system. In *Proceedings of 25th International Conference on Very Large Data Bases*, M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik, and M. L. Brodie, Eds. (Edinburgh, Scotland). Morgan-Kaufmann, San Francisco, Calif., 471–482.

LAKSHMANAN, L. V. S., SADRI, F., AND SUBRAMANIAN, S. N. 2001. SchemaSQL—An extension to SQL for multidatabase interoperability. *ACM Trans. Datab. Syst. 26*, 4 (Dec.), 476–519.

LENZERINI, M. 2002. Data integration: A theoretical perspective. In *Proceedings of the 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, L. Popa, Ed. (Madison, Wisc., June 3–5). ACM, New York, 233–246.

LITWIN, W., ABDELLATIF, A., ZEROUAL, A., AND NICOLAS, B. 1989. MSQL: A multidatabase language. *Inf. Sci. 49*, 59–101.

MASERMANN, U. AND VOSSEN, G. 2000. SISQL: Schema-independent database querying (on and off the web). In *Proceedings of the International Database Engineering and Applications Symposium* (*IDEAS'00*) (Yokohama, Japan). IEEE Computer Society Press, Los Alamitos, Calif., 55–65.

NAQVI, S. A. AND TSUR, S. 1989. *A Logical Language for Data and Knowledge Bases*. Computer Science Press, Inc., New York, N.Y.

PAPAKONSTANTINOU, Y., GARCÍA-MOLINA, H., AND ULLMAN, J. 1996. MedMaker: A mediation system based on declarative specifications. In *Proceedings of the 12th International Conference on Data Engineering* (New Orleans, La.). IEEE Computer Society Press, Los Alamitos, Calif.

RAMAKRISHNAN, R. AND GEHRKE, J. 2003. *Database Management Systems*, Third ed. McGraw-Hill, New York.

ROBERTSON, E. L. AND WYSS, C. M. 2004. Optimal tuple merge is NP-Complete. Tech. Rep. TR599, Indiana University Computer Science. July.

ROOD, C. M., VAN GUCHT, D., AND WYSS, F. I. 1999. MD-SQL: A language for meta-data queries over relational databases. Tech. Rep. TR-528, Indiana University at Bloomington. July.

ROSS, K. A. 1992. Relations with relation names as arguments: Algebra and calculus. In *Proceedings of the 11th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. (San Diego, Calif.). ACM, New York, 346–353.

SALTOR, F., CASTELLANOS, M., AND GARCIA-SOLACO, M. 1993. Overcoming schematic discrepancies in interoperable databases. In *DS-5*, D. K. Hsiao, E. J. Neuhold, and R. Sacks-Davis, Eds. IFIP Transactions, vol. A-25. North-Holland, Amsterdam, The Netherland, 191–205.

SU, H., CLAYPOOL, K., AND RUNDENSTEINER, E. A. 2000. Extending the object query language for transparent metadata access. Tech. Rep. WPI-CS-TR-00-19, Worcester Polytechnic Institute.

VAN DEN BUSSCHE, J., VAN GUCHT, D., AND VOSSEN, G. 1993. Reflective programming in the relational algebra. In *Proceedings of the 12th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (Washington, D.C., May 25–28). ACM, New York, 17–25.

VAN DEN BUSSCHE, J., VANSUMMEREN, S., AND VOSSEN, G. 2004. Meta-SQL: Towards practical meta-querying. In *Advances in Database Technology (EDBT 2004): 9th International Conference on Extending Database Technology* (Heraklion, Crete, Greece). Springer-Verlag, New York, 823–825.

VAN DEN BUSSCHE, J. AND WALLER, E. 2002. Polymorphic type inference for the relational algebra. *J. Comput. Syst. Sci. 64*, 3, 694–718.

WYSS, C. AND VAN GUCHT, D. 2001. A relational algebra for data/metadata integration in a federated database system. In *Proceedings of the 2001 ACM CIKM International Conference on Information and Knowledge Management* (Atlanta, Ca., Nov. 5–10). ACM, New York, 65–72.

WYSS, C., WYSS, F., AND VAN GUCHT, D. 2001. Augmenting SQL with dynamic restructuring to support interoperability in a relational federation. In *Engineering Federated Information Systems, Proceedings of the 4th Workshop* (*EFIS 2001*), R.-D. Kutsche, S. Conrad, and W. Hasselbring, Eds. (Berlin, Germany Oct. 9–10). IOS Press, 5–18.

WYSS, C. M. 2002. Relational interoperability. Ph.D. dissertation, Indiana University at Bloomington.

Received October 2003; revised September and December 2004; accepted February 2005