

Extending Relational Query Optimization to Dynamic Schemas for Information Integration in Multidatabases

Catharine M. Wyss
Indiana University
cmw@cs.indiana.edu

Felix I. Wyss
Interactive Intelligence, Inc.
felixw@inin.com

ABSTRACT

This paper extends relational processing and optimization to the FISQL/FIRA languages for dynamic schema queries over multidatabases. Dynamic schema queries involve the creation and restructuring of metadata at runtime. We present a full implementation of a FISQL/FIRA engine, which includes subqueries and all transformational capabilities of FISQL/FIRA on distributed, multidatabase platforms. An important application of the system is to enhance traditional information architectures by enabling the creation and maintenance of dynamic wrappers and mapping queries at source databases within GAV, LAV, GLAV, peer-to-peer, or other integration frameworks. In addition to fully supporting FISQL/FIRA on multidatabases, our implementation introduces a bi-level optimization paradigm where purely relational sub-fragments of queries are pushed into source engines. This paradigm shares features of canonical distributed database processing, but has a new dimension through the extension of the relational model to dynamic schemas. We present empirical results showing the feasibility of optimization in this context, and discuss tradeoffs involved. Our system is the first to extend relational databases with these capabilities on this scale.

Categories and Subject Descriptors

H.2.3 [Database Management]: Languages—*Query languages*; H.2.4 [Database Management]: Systems—*Query processing*

General Terms

Languages

Keywords

Data integration, dynamic schema query processing, federated databases, federated data model, FIRA, FISQL, federated interoperable RA, federated interoperable SQL, metadata integration, metadata query processing.

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

SIGMOD'07, June 12–14, 2007, Beijing, China.

Copyright 2007 ACM 978-1-59593-686-8/07/0006 ...\$5.00.

1. INTRODUCTION

Data integration is a perennial issue in database management, and continues to receive much attention in recent years [13, 14]. Although the relational model is widespread and an elegant data storage paradigm, to date it has lacked features necessary for large scale data integration [20]. In particular, a problem that plagues data integration systems is that of *dynamic schemas*, such as those in our example federation of Figure 1. Until recently, the relational model has lacked the *transformational completeness* to express the data-metadata restructuring inherent in such applications.

Figure 1 shows a loosely coupled federation of databases for a retail supply network. In this federation, there is no single, global schema, and source databases are updated independently. As shown, relation ORDERS of the CA database contains aggregated total sale information in a normalized relational format. On the other hand, relation ORDERS of the IL database shows semantically equivalent data in a “spreadsheet” format, where each customer occupies a separate column.¹ In database NY, a separate relation is used for each customer. Note that in this example there is overlap among the databases, for example the same information about customer SMITH is represented in all three databases. On the other hand, customer MILLS is only represented in the IL database.

Our goal was to implement a relational database system that could effectively and dynamically query data and metadata in federations such as Figure 1. Recently, we developed a relational language for dynamic schema querying, *Federated Interoperable SQL* (FISQL) [20]. The main advantage of FISQL in terms of query processing and optimization purposes is that it has an equivalent algebra, *Federated Interoperable RA* (FIRA), that is a direct extension of canonical RA. In particular, FIRA inherits the algebraic properties of RA that are essential for query optimization, such as commutativity, associativity, and distributivity. Furthermore, FIRA contains additional operators for data-metadata transformations that are more general and widely applicable than predecessor languages [20].

1.1 Main Contributions

Our contributions in this paper are as follows.

- We refine the semantics and syntax of FISQL/FIRA based on our implementation by introducing a *no-*

¹Although from a relational standpoint this design is highly inefficient, in practice many users employ spreadsheet programs to store data, resulting in such designs when naïvely imported to RDBMS.

CA ORDERS			IL ORDERS					
CUSTKEY	DATE	TOTALSALE	DATE	SMITH	HENRY	JAMES	MILLS	BURNS
SMITH	2005-08-01	955	2005-08-01	967	⊥	689	957	68
SMITH	2005-08-02	589	2005-08-02	566	953	18	45	⊥
JONES	2005-08-01	938						
BROWN	2005-08-01	23						
BROWN	2005-08-02	55						
BLACK	2005-08-01	999						
BLACK	2005-08-02	300						
HENRY	2005-08-02	935						

NY SMITH		BROWN		JAMES	
DATE	TOTALSALE	DATE	TOTALSALE	DATE	TOTALSALE
2005-08-01	957	2005-08-01	230	2005-08-01	699
2005-08-02	532	2005-08-02	551	2005-08-02	81

BURNS		MILLS		WEBER	
DATE	TOTALSALE	DATE	TOTALSALE	DATE	TOTALSALE
2005-08-01	86	2005-08-01	866	2005-08-01	908
2005-08-02	978	2005-08-02	595	2005-08-02	139

Figure 1: Multidatabase containing information on customer orders.

overwrite semantics for FIRA metadata operators, and employing this to improve the presentation of FIRA by obviating the need for meta-metavariables [20].

- We describe a modular, non-intrusive implementation of a FISQL/FIRA query engine (Figure 10). Our engine supports full dynamic-schema FISQL, including sub-queries and multi-source input.
- We introduce novel, generalized FIRA rules for rewriting dynamic schema queries and subqueries throughout Section 3. We organize these into two optimization phases of FIRA plan rewrites for equivalent FISQL.
- We introduce a *bi-level* optimization paradigm for exploiting source relational query optimizers (Section 2.3). This approach shares some features of canonical distributed database query processing [19], however innovations were required for handling dynamic schemas.
- We give experimental results showing the feasibility of extended relational optimization using FIRA. Our work illustrates that FIRA can form the basis for cost-based dynamic schema optimization of FISQL queries. Furthermore, we show that FIRA *write-out* strategies are likely to perform better than *unwinding* FISQL queries using system catalogs. The latter approach has appeared in previous work on SchemaSQL [1, 11]; here, we formalize the approach and discuss its weaknesses.

An important application of our work is to facilitate data integration in frameworks incorporating mappings between source databases. In particular, our work can facilitate GAV, LAV, GLAV [13], BAV [16], as well as peer-to-peer approaches, by extending source RDBMS with dynamic schema capabilities so that each becomes a *unit of interoperability* more robust under schema transformation and evolution. Adding the flexibility and transformation space of FISQL to source engines would give broader support for the mappings that are crucial for integration frameworks. A caveat is that our work focuses on *structural* integration, whereas

semantic reconciliation remains one of the toughest problems facing data integration [14]. However, researchers have addressed this with some success, and our framework complements systems such as Clio [17] or iMAP [4].

Finally, we give a brief note on terminology. The term *multidatabase* [15] has come to mean a collection of (relational) databases containing semantically similar but possibly structurally dissimilar information. On the other hand, the term *federated* sometimes carries the connotation of a common global schema [18]. In this paper, we use the terms “federation” and “multidatabase” interchangeably to mean a collection of databases containing semantically related information, and make no commitment to any overarching global schema or network architecture.

2. FISQL/FIRA

For this paper, we assume an underlying domain of atomic elements, denoted Ω . Elements of Ω are assumed to be alphanumeric strings and will be written in all-caps, as in Figure 1. Our implementation currently supports both string and integer types, and in fact even allows both to occur as column or relation names,² or together in a single column. However, this extended typing is orthogonal to FIRA optimization, so a discussion is omitted here. We assume distinct elements “ $\perp$ ” (the null value), where $\perp \notin \Omega$, and also $\epsilon \in \Omega$ (the empty string).

One of the main features of the federated data model underpinning FISQL and FIRA is that *both data and metadata are taken from Ω* . This seems natural, since metadata is usually thought of as strings, and enables us to formally compare data and metadata within the model.

Both FISQL and FIRA include the ability to create and restructure relational schemas dynamically at run time. In order to support this, [20] defines a broader notion of *schema*

²Note that we allow any atomic value (including numbers) to be promoted to column headings or relation names. Although this seems strange at first, it is a natural consequence of dynamic schema creation. For example, in spreadsheet data, column headings such as dates or quarters are heavily used.

within an extended relational data model known as the *federated data model*. Here, we assume the following definitions.

Definition 2.1

1. A (federated) *tuple*, t , is a mapping $t : \Omega \rightarrow \Omega \cup \{\perp\}$. We use $t[A]$ for $A \in \Omega$ to indicate the image of A under t . Every tuple maps all but a *finite* number of elements in Ω to $\perp$. The *active schema* of t , denoted $asch(t)$, is the non-null mapped input elements.
2. A (federated) *relation*, r , consists of a name (denoted $name(r) \in \Omega$), and a body (denoted $body(r)$). The relation body is a finite set of (federated) tuples. The *active schema* of r , denoted $asch(r)$, is $\cup_{t \in r} asch(t)$.
3. A (federated) *database*, d , is a finite set of (federated) relations.
4. A *federation* is a finite set of (federated) databases.

Note that the “full” schema of every input relation is all of Ω . One consequence is that relation schemas always match; hence FIRA versions of $\times$, $\cup$ and $-$ are always “outer” (see Section 2.2.1). The federated data model is robust under dynamic schema creation, which is its main appeal [20]. By convention, a federated relation is finitely represented as a table containing columns with at least one non-null value.

2.1 FISQL

A full EBNF for FISQL as we refined it from [20] is given in Appendix A. *Metavariables*, which range over attribute or relation names, are declared using a “:” in the **from** clause. When these attribute or relation names are fixed, the specific value may be substituted in the **from** clause and signified with a “.” instead of a “:”.³ A full presentation of FISQL is available in [20]. The following examples summarize the relevant syntactic features.

Example 2.1 (IL $\rightarrow$ CA)

The following query shows the FISQL syntax for declaring and using an attribute metavariable. This is the transformation from the IL to the CA database.

```
select Iatt as 'CUSTKEY',
       I.DATE as 'DATE',
       I.Iatt as 'TOTALSALE'
into 'NewIL'
from IL.ORDERS:Iatt as I
where Iatt <> 'DATE'
```

Example 2.2 (NY $\rightarrow$ CA)

The following query shows the FISQL syntax for declaring and using a relation metavariable. This is the transformation from the NY to the CA database.

```
select Nrel as 'CUSTKEY',
       N.DATE as 'DATE',
       N.TOTALSALE as 'TOTALSALE'
into 'NewNY'
from NY:Nrel as N
```

Example 2.3 (CA, IL Comparison with Subquery)

The following query finds customers who spent more than \$950 on the same date according to both the CA and IL

³The “.” notation is shorthand for a “:” declaration in the **from** clause followed by an equality constraint in the **where** clause.

databases. Note that the subquery uses the FISQL “on” keyword for dynamic attribute creation.⁴

```
select Iatt as 'CUSTKEY'
into 'RESULT'
from (select T.DATE as 'DATE',
           T.TOTALSALE on T.CUSTKEY
       into 'NewCA'
       from CA.ORDERS as T
       ).NewCA:Catt as C,
   IL.ORDERS:Iatt as I
where I.DATE = C.DATE and
      Iatt = Catt
      I.Iatt > 950 and
      C.Catt > 950
```

This query can be phrased without the subquery. However, a human user might prefer to use a more basic subquery to first perform the intended restructuring and then do the comparison in the outer query, which makes the overall query easier to understand. A strength of our implementation is that even if the input query is inherently inefficient in this way, our optimization techniques will automatically reduce it to a much simpler form (see Example 3.1).

Example 2.4 (CA, NY Comparison with Subquery)

Similarly, the following query employs the transformation from the NY database to the CA database to find customers who have spent more than \$950 on the same date according to both those databases. Note that the inner query dynamically creates a database as the term to the right of the **into** keyword dynamically resolves to the CUSTKEY values.

```
select Nrel as 'CUSTKEY'
into 'RESULT'
from (select T.DATE, T.TOTALSALE
       into T.CUSTKEY
       from CA.ORDERS as T
       ):Crel as C,
   NY:Nrel as N
where N.DATE = C.DATE and
      Nrel = Crel and
      N.TOTALSALE > 950 and
      C.TOTALSALE > 950
```

2.2 FIRA

In this section we present a brief summary of FIRA and discuss how FISQL input queries are translated to FIRA query plans in our implementation. Note that our semantics of some FIRA operators differs from the presentation in [20]. In particular, we recognized that for some operators, either a “no-overwrite” or an “overwrite” semantics could be consistent for FIRA. In the implementation, it became necessary to choose one of these semantics. The one that facilitates query optimization best is a *no-overwrite* semantics. This is detailed in the presentation below.

Also, the presentation in [20] requires a notion of *meta-metadata*. With the no-overwrite semantics in place, we can remove this additional complexity and instead use ordinary metadata, as detailed below.

Like FISQL, a FIRA query takes databases as input and returns a single database. This closure is a natural parallel of SQL/RA closure since SQL and RA take relations as input and return a single relation.

⁴To exactly reproduce the format of the IL.ORDERS relation in the subquery, an operation to appropriately merge tuples with null values would be required. Adding a *merge* operation is problematic for several reasons, but can be done successfully [21]. For many useful comparisons a diagonal transpose suffices (as here).

2.2.1 Relational Core of FIRA

FIRA contains federated versions of the RA operators extended to multidatabases, which we denote using “hats” (following [20]). One of the fundamental properties of the FISQL/FIRA framework is that RA is equivalent to a sub-algebra of FIRA (namely, the algebra delineated by the federated RA counterparts). For the purposes of this paper, this means RA algorithms and equivalences carry over directly to FIRA. This enables us to elegantly use and extend the considerable body of research on canonical relational query optimization.

In the following definitions, we will use the notation $\langle n, x \rangle$ where $n \in \Omega$ and x is a set of (federated) tuples to mean the federated relation with name n and body x .

For the unary RA operators, Π and σ , the federated counterparts simply map the relational operations to the bodies of constituent relations.⁵

Definition 2.2 (Unary Federated RA Operations)

Let d be a (federated) database instance, θ be a selection condition, and $X \subseteq \Omega$ a finite set of atoms.

1. (Federated Selection)

$$\hat{\sigma}_\theta(d) = \{\langle name(r), \sigma_\theta(body(r)) \rangle \mid r \in d\}.$$

2. (Federated Projection)

$$\hat{\Pi}_X(d) = \{\langle name(r), \Pi_X(body(r)) \rangle \mid r \in d\}.$$

Note that for the federated model, the columns in a projection have not “disappeared” as in the canonical model; rather, the effect of a project is to set all values in non-projected columns to $\perp$.

In the examples below, we will use a convenient shorthand which combines projection and renaming. The notation $\Pi_A^B(r)$ means that r is first projected on A , and then the A column is renamed to B (where $A, B \in \Omega$ and $B \notin asch(r)$).

For the binary RA operators, $\times$, $\cup$, and $-$, non-trivial generalizations are needed. The federated counterparts for these operators only combine relations from the input databases where those relations *have the same name*. This subtle but important divergence from previous multidatabase operators such as MRA [7] preserves important properties such as commutativity of $\bowtie$.

Definition 2.3 (Binary Federated RA Operations)

Let d_1 and d_2 be two (federated) database instances.

1. (Federated Cartesian Product)

$$\begin{aligned} d_1 \hat{\times} d_2 &= \{\langle name(r_1), body(r_1) \times body(r_2) \rangle \mid \\ &\quad r_1 \in d_1, r_2 \in d_2 \text{ and } name(r_1) = name(r_2) \}. \end{aligned}$$

2. (Federated Set Union)

$$\begin{aligned} d_1 \hat{\cup} d_2 &= \{\langle name(r_1), body(r_1) \cup body(r_2) \rangle \mid \\ &\quad r_1 \in d_1, r_2 \in d_2 \text{ and } name(r_1) = name(r_2) \} \\ &\cup \{r_1 \in d_1 \mid \nexists r_2 \in d_2. name(r_2) = name(r_1)\} \\ &\cup \{r_2 \in d_2 \mid \nexists r_1 \in d_1. name(r_1) = name(r_2)\}. \end{aligned}$$

⁵FIRA includes an extended renaming operation, $\hat{\rho}$. For conciseness we do not include this operation here.

3. (Federated Set Difference)

$$\begin{aligned} d_1 \hat{-} d_2 &= \{\langle name(r_1), body(r_1) - body(r_2) \rangle \mid \\ &\quad r_1 \in d_1, r_2 \in d_2 \text{ and } name(r_1) = name(r_2) \} \\ &\cup \{r_1 \in d_1 \mid \nexists r_2 \in d_2. name(r_2) = name(r_1)\}. \end{aligned}$$

As mentioned, union or difference operations always succeed on federated relations, since the full schema of these relations is always Ω . These operations are *outer* in that columns from one relation not appearing in the other are assumed to exist and contain only null values. For examples and further discussion of the relational core of FIRA, please see [20].

2.2.2 Extended Relational Operators

In addition to the relational core, FIRA contains six new operators for creating and manipulating dynamic schemas.

Metadata Demotion. For a (federated) relation, r , and $A, B \in \Omega$ let $metadata_A^B(r)$ be the relation with schema $\{A, B\}$ containing a tuple of the form $\langle A : name(r), B : a \rangle$ for each $a \in asch(r)$.

Definition 2.4 (Metadata Demotion)

Let d be a (federated) database instance and $A, B \in \Omega$. Then $\downarrow_A^B(d)$, the *metadata demotion* over d is:

$$\downarrow_A^B(d) = \{\langle name(r), metadata_A^B(r) \times body(r) \rangle \mid r \in d\}.$$

No-Overwrite Semantics: In the case where A or B already exists in the relation being paired with its metadata, we assume the operation has *no effect* on that column. In other words, previously existing columns in r are *not* overwritten in a metadata demotion.

In the FIRA query plans for our examples, we will use two distinct versions of the $\downarrow$ operator. $\downarrow_A$ is assumed to pair only the attribute names with the relation body. $\downarrow^B$ is assumed to pair only the relation name with its body.

Drop Projection. In addition to ordinary projection, the dynamic nature of federated schemas necessitates adding a *Drop Projection* to FIRA, denoted Π , and defined as follows.

Definition 2.5 (Drop Projection)

Let d be a (federated) database and $A \in \Omega$.

$$\Pi_A(d) = \{\langle name(r), \Pi_{asch(r) - \{A\}}(r) \rangle \mid r \in d\}.$$

Dynamic Attributes. FIRA contains an operator for adding attributes and associated contents dynamically to a relation (τ). In addition, FIRA includes an operator for *dereferencing* a column of values, which is effectively a projection that can project a different column for each tuple (Δ).

Definition 2.6 (Diagonal Transpose)

1. Let r be a (federated) relation and $A, B \in \Omega$. Then the *Diagonal Transpose* of r , denoted $\tau_A^B(r)$, is the set of tuples t' where for each $t \in r$ and for each $C \in \Omega$, t' is constructed as follows.

$$t'[C] = \begin{cases} t[A] & \text{when } C = t[B] \text{ and } C \notin asch(r), \\ t[C] & \text{otherwise.} \end{cases}$$

CUSTKEY	DATE	TOTALSALE	SMITH	JONES	BROWN	BLACK	HENRY
SMITH	2005-08-01	955	955	⊥	⊥	⊥	⊥
SMITH	2005-08-02	589	589	⊥	⊥	⊥	⊥
JONES	2005-08-01	938	⊥	938	⊥	⊥	⊥
BROWN	2005-08-01	23	⊥	⊥	23	⊥	⊥
BROWN	2005-08-02	55	⊥	⊥	55	⊥	⊥
BLACK	2005-08-01	999	⊥	⊥	⊥	999	⊥
BLACK	2005-08-02	300	⊥	⊥	⊥	300	⊥
HENRY	2005-08-02	935	⊥	⊥	⊥	⊥	935

(a)

DATE	SMITH	HENRY	JAMES	MILLS	BURNS	NEW1	NEW2
2005-08-01	967	⊥	689	957	68	DATE	2005-08-01
2005-08-01	967	⊥	689	957	68	SMITH	967
2005-08-01	967	⊥	689	957	68	HENRY	⊥
2005-08-01	967	⊥	689	957	68	JAMES	689
2005-08-01	967	⊥	689	957	68	MILLS	957
2005-08-01	967	⊥	689	957	68	BURNS	68
2005-08-02	566	953	18	45	⊥	DATE	2005-08-02
2005-08-02	566	953	18	45	⊥	SMITH	566
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

(b)

Figure 2: (a) $\tau_{\text{TOTALSALE}}^{\text{CUSTKEY}}(\text{CA.ORDER})$, (b) $\Delta_{\text{NEW1}}^{\text{NEW2}}(\downarrow_{\text{NEW1}}(\text{IL.ORDER}))$

2. For a (federated) database, d ,

$$\tau_A^B(d) = \{\langle \text{name}(r), \tau_A^B(r) \rangle \mid r \in d\}.$$

Note the *no-overwrite* semantics for transposition: if any values being promoted already exist in the active schema, the corresponding columns are left untouched.

Example 2.5

Figure 2 (a) shows the result of a diagonal transpose on the **CA.ORDER** table.

The operator τ has affinities with the PIVOT operator available in commercial RDBMS and spreadsheet applications [2]. In fact, PIVOT is basically a transpose combined with an *optimal tuple merge* operator, μ [21].

Definition 2.7 (Dereference Projection)

- Let r be a (federated) relation and $A, B \in \Omega$. Then the *Dereference Projection* of r , denoted $\Delta_A^B(r)$, is the set of tuples t' where for each $t \in r$ and for each $C \in \Omega$, t' is constructed as follows.

$$t'[C] = \begin{cases} t[t[A]] & \text{when } C = B \text{ and } B \notin \text{asch}(r), \\ t[C] & \text{otherwise.} \end{cases}$$

- For a (federated) database, d ,

$$\Delta_A^B(d) = \{\langle \text{name}(r), \Delta_A^B(r) \rangle \mid r \in d\}.$$

Once again, note the *no-overwrite* semantics for Δ .

Example 2.6

Figure 2 (b) shows the result of an attribute demotion followed by a dereference projection on **IL.ORDER**. Note that a dereference projection may result in columns with heterogeneous atomic types, a feature which we support.

Including Dereference in Selection and Projection. A convenient shorthand was introduced in [20] for including dereference columns directly in selection conditions and projection lists. This is the “overbar” notation. For example, for $A, B \in \Omega$ and r a (federated) relation instance, the term $\sigma_{\overline{A}=B}(r)$ means we first dereference the column A , and then compare the resulting new column to B . In other words, this term is equivalent to $\sigma_{C=B}(\Delta_A^C(r))$ where $C \in \Omega$ and $C \notin \text{asch}(r)$. Similarly, the “overbar” notation may be used in a projection, where $\Pi_{\overline{A}}^B(r)$ means $\Pi_C^B(\Delta_A^C(r))$. Note that in such projection terms the atom B must be included in the superscript, fixing the renaming that occurs.

Not only is the overbar notation convenient for the translation from FISQL to FIRA, but in fact “ $\sigma_{\overline{A}=B}(r)$ ” and “ $\Pi_{\overline{A}}^B(r)$ ” have separate implementation algorithms which are more efficient than a Δ followed by a σ or Π . This is similar to the manner in which a $\bowtie$ operation is more efficient than a $\times$ followed by a σ (see Section 3).

Dynamic Relations. Finally, FIRA contains operations for splitting a single relation into several, and also for unioning several into one, as follows.

Definition 2.8 (Partition a Relation)

- Let r be a (federated) relation and $A \in \Omega$. The result of partitioning r on A , denoted $\wp_A(r)$, is a database obtained as follows.

$$\wp_A(r) = \{\langle a, \sigma_{A=a}(\text{body}(r)) \rangle \mid \exists t \in \text{body}(r). t[A] = a\}.$$

- For a (federated) database, d ,

$$\wp_A(d) = \bigcup_{r \in d} \wp_A(r).$$

Definition 2.9 (Generalized Union)

Let d be a (federated) database and recall $\epsilon \in \Omega$ is the “empty name”. The *generalized union* of d , denoted $\Sigma(d)$, is

That is, $\Sigma(d)$ is a database with a single relation whose name is ϵ and body is the (federated) union of the bodies of every relation in d .

In the “named” RA, a Cartesian product technically fails when the input relations have overlapping schemas. FIRA inherits this behavior. However, in practise, SQL allows *scoping* of the columns in input relations using the **as** keyword in the **from** clause. This translates to an “entire relation rename” in RA. FISQL similarly allows this type of automatic renaming, or scoping. Our implementation does include a notion of the *scope* of a tuple or metavariable which codifies this idea. However, the range of a FISQL tuple variable is naturally seen as a *database*, not a relation. To remain faithful to this idea in our example query plans, we will subscript Σ operations with a tuple variable, which gives their scope in accordance with the FISQL to FIRA translation.

In the translation from FISQL to FIRA, intermediate results may have new (temporary) columns as a result of the decomposition into FIRA operators. These are supported in the implementation and employ a separate namespace to avoid collisions. In FIRA query plans (for example Figure 3, below), system-generated identifiers will be prefixed with double underscores (for example “`__latt`”) to distinguish them from identifiers appearing in the input query. Note that the appearance of such identifiers is an artifact of the translation from FISQL to FIRA, and does *not* add expressive power to FIRA itself.

One of our goals in processing and optimizing FISQL queries is to take advantage as much as possible of source query engines, which will likely include highly sophisticated query optimizers. This goal forms the basis of our *bi-level* approach to FIRA query optimization. In a FIRA query plan, purely relational operations can be pushed toward the leaves (where source databases come into play). In this case, we can bundle these (purely relational) operations into our request for data from the source. This paradigm is reified in the physical source interface operator Φ .

Let D be a (relational) database schema consisting of relation schemas $R_1, \dots, R_n$, and d an instance of D . Let $Q_1, \dots, Q_n$ be RA queries over $R_1, \dots, R_n$ (respectively). Then we *federate* d using the Φ operator, defined as follows.

Here, we overload R_i to mean the *name* of the source relation being returned. Thus, Φ applies a filter query to source relations and also pairs them with their name as given by source system tables. Additionally, the filter query may have the form “select * from *”. The meaning of this is to return all relations in d , with no additional relational filtering.

to a relational union operation, since result schemas are allowed to vary dynamically. Some care must be taken to respect scope and naming issues when pushing relational query fragments into Φ and combining filter queries, due to the dynamic schema capabilities of FIRA. Using FIRA as a vehicle, we have extended the canonical “pushing” of filter queries in distributed databases to dynamic schemas.

A naïve FIRA query plan for the FISQL query in Example 2.3 is shown in Figure 3. In fact, this translation is *semi-naïve* in that intermediate projections and/or drop projections have been omitted to make the result easier to visualize. Note that leaves contain Φ operations. In the next section, we will show how most of the complexity of such queries can be absorbed into Φ .



A semi-naïve FIRA query plan for the the FISQL query in Example 2.4 is shown in Figure 4.

Our optimizer utilizes a “two-phase” rewriting scheme. The first phase follows the fundamental techniques of canonical relational optimization, but rewrite rules are augmented significantly with support for FIRA operations as well as RA. In phase I, $\hat{\sigma}$, Π , and Π constraints are rewritten to reduce the size of intermediate expressions. Intuitively, this phase “pushes” constraining expressions as far down the FIRA query plan tree as possible, including (where possible), into Φ expressions. Furthermore, selection conditions and Cartesian products are combined into $\bowtie$ operations. This phase of the optimization is detailed in Section 3.1.

478

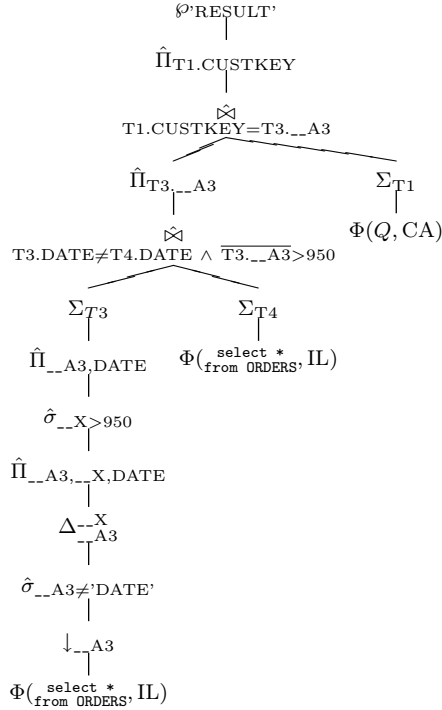


Figure 7: Fully constrained plan for Example 3.3

Example 3.4

Figure 9 shows the result of unwinding the query in Example 3.3, as an alternative to the plan of Figure 8.

When both the write-out and unwind techniques apply, there is the question of which to apply (or whether to employ a combination of both techniques). The write-out technique is robust as the number of attributes and/or relations in the source catalog grows. In contrast, metavariable unwinding can result in very large union queries, especially if there are several metavariables declared in the **from** clause. We investigate empirical performance of the two approaches in Section 4.2.1. and present techniques for dealing with large union queries in Φ in Section 4.2.2.

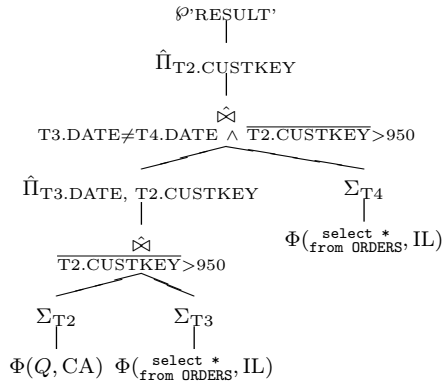
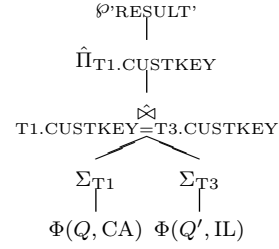


Figure 8: Metavariable-free plan for Example 3.3



Query Q' issued to database IL is:

```
select 'SMITH' as 'CUSTKEY'
from   ORDERS as 01, ORDERS as 02
where  01.SMITH > 950 and
       02.SMITH > 950 and
       01.DATE <> 02.DATE

union
...
union
select 'BURNS' as 'CUSTKEY'
from   ORDERS as 01, ORDERS as 02
where  01.BURNS > 950 and
       02.BURNS > 950 and
       01.DATE <> 02.DATE
```

Figure 9: Unwound plan for Example 3.3

4. EMPIRICAL RESULTS

4.1 System Architecture

Figure 10 depicts the architecture of our FISQL/FIRA engine. We currently have an implementation written in C# using the Microsoft .NET framework [22]. Module (1) uses ANTLR [23] to assist with parsing and translation. Module (5) supports connections to ODBC sources.

4.2 Experiments

To evaluate the impact of our optimization heuristics, tests were run on data similar to that in Figure 1. Note that the ORDERS federation is inspired by TPC-H benchmark data. However, TPC-H specifies fixed-schema relations, and for our tests it was important to generate input databases with varying schemas. Metrics for generating the data were as follows.

- The number of customers varied from 25 to 1000.⁶ Customer names were chosen from recent census data, assuming a 10% overlap of names between sources.
- Values for TOTALSALE were randomly distributed in the range 0 to 1000. Additionally, a random sample of 10% of customers were assumed to buy something on any given date. If a customer did not make a purchase on a date, the TOTALSALE is $\perp$.
- Data was generated for 1800 consecutive dates (5 years). The range is the same for all databases.

Note that each of the two databases involved in the query were generated independently, and re-created for each test to thwart query plan caching by source engines, which can

⁶Note that the upper limit on the number of customers comes from SQLServer, which supports a maximum of 1024 columns in the IL.ORDERS relation.

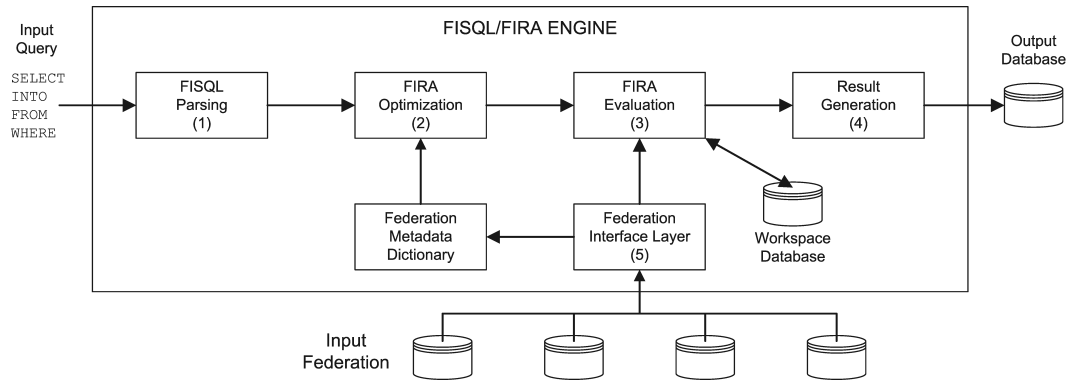


Figure 10: FISQL Processor Architecture

reduce times for the Φ operation by an order of magnitude or more (and so would skew our results).

Tests were run on a system with a 2.8 GHz Pentium D processor and 4 GB RAM running Windows XP (SP2). All databases are hosted on a local SQL Server 2005 (Developer Edition) [24]. Our implementation is in C# using the .NET Framework 2.0 [22].

4.2.1 Relative Optimization Performance

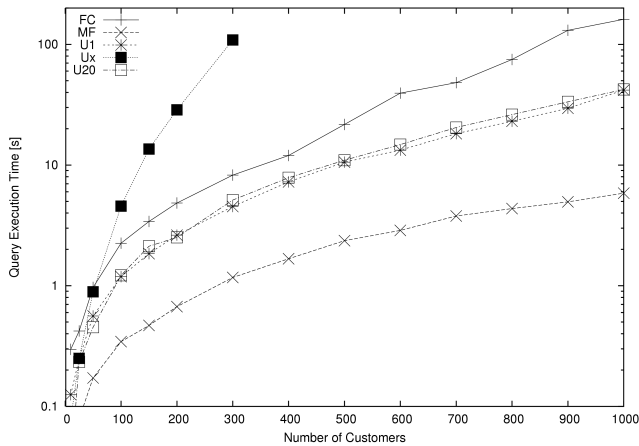


Figure 11: Relative impact of FIRA optimizations on query from Example 3.3. FC = fully constrained plan (Figure 7). MF = metavariable-free plan (Figure 8). Others are unwound plans of varying union batch size (see text for details).

Figure 11 shows query processing times for different FIRA plans for the FISQL query from Example 3.3. Times are shown in seconds for the fully constrained plan from Figure 7 (FC), the metavariable-free plan from Figure 8 (MF), and three versions of the unwound plan from Figure 9 (U1, Ux, U20). The unwound plans differ in the number of unions in the filter query sent by the Φ operator to source database IL as one *batch*. In U1, a single unwound instantiation is sent per batch. In Ux, the filter query is the union of *all* unwound instantiations (so there is a single batch).⁷ In U20,

⁷On batches with more than 300 unions, SQLServer reported that

each filter query contains the union of 20 unwound instantiations. Larger batches mean fewer queries submitted to source engines and thus fewer network round trips; on the other hand, larger batches can take disproportionately longer to process in the source databases. This is discussed further below (Section 4.2.2).

Our implementation accepts *any* FISQL query adhering to the EBNF in Figure 13. Although tests were run for several query types, the query in Example 3.3 is complex enough to showcase our optimizations and also exhibits representative behavior.

The fully constrained plan (FC in Figure 11), which still includes FIRA $\downarrow$ operations, is considerably slower than the $\downarrow$ -free plans. For 1000 customers, the evaluation of the FC plan includes as its largest intermediate result a relation of about 1.8 million tuples. In contrast, the metavariable-free plan (MF in Figure 11) computes intermediate results of 9000 or fewer tuples. Previous implementations of dynamic schema queries focused largely on unwinding as an evaluation strategy [1, 11]. Our results show that metavariable-free plans perform better in practice. This highlights the advantage of an approach based on an underpinning, formal algebra, rather than ad-hoc processing based on system metadata. Another advantage of plans that are *not* unwound is that they are independent of the metadata in source databases. For example, plan caching of unwound plans will be complicated by changes to source metadata, whereas metavariable-free or fully constrained plans will be truly schema independent in this sense.

4.2.2 Effect of Batching on Filter Queries

Figure 12 shows how the overall runtime of Φ varies with the size of filter query batches. Input databases were as in the previous results, fixed for 1000 customers. Along the x-axis, a batch size of “N” means that each filter query contained N unions (arising from metavariable instantiations). The results of these filter queries are combined within Φ itself in our evaluation engine to produce the result. Overall times are shown for the Φ operation on the IL database according to a fully unwound plan for the query in Example 2.3 (IL), as well as for the Φ operation on the NY database according to a fully unwound plan for the query in Example 2.4. Processing times for other FISQL queries showed similar trends according to filter query batch size.

there was insufficient system memory to run the query.

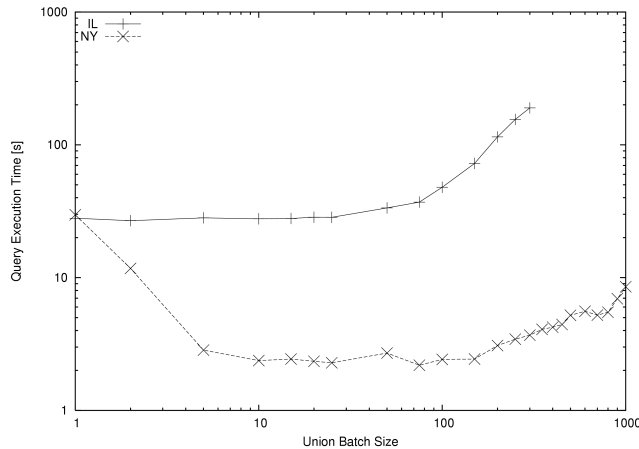


Figure 12: Effect of union batch size on unwound versions of queries from Examples 2.3 and 2.4.

Since we ran tests where sources resided locally, network latency was not an issue. In general, network latency in a distributed setting needs to be taken into consideration when determining the optimal batch size for filter queries.

Interestingly, SQLServer performs well on the *fixed schema* unions arising from the NY database, even when the overall number of unions in the filter query grows. In contrast, for the *dynamic schema* case of the IL.ORDERS table, the SQLServer query engine runs out of memory on filter queries involving more than about 300 unions.

5. RELATED WORK

To date, very little formalism has been developed to handle dynamic schemas in SQL and RA. FISQL/FIRA is the first set of languages to give a transformationally complete, fully compositional semantics to dynamic relational schemas [20]. The tabular algebra has done this for an extended data model (the tabular data model) [8]. Although not currently implemented (as far as we know), the algebra is capable of elegantly simulating *all* spreadsheet transformations [8].

Our approach shares some of the features of canonical distributed query processing [19]. In addition, predecessor integration systems such as Garlic [9] employed similar techniques for pushing queries into source databases. Our work differs fundamentally from systems such as Garlic or TSIMMIS [6] in that (1) we extend the canonical relational model instead of employing complex or object-oriented structures to accomplish integration and (2) our work focuses on the dynamic data $\leftrightarrow$ metadata transformations inherently arising in integration but not widely studied.

An important predecessor of FISQL is SchemaSQL [11, 12]. FISQL diverges from SchemaSQL in both syntax and semantics in important ways [20]. As an example, the FISQL **from** clause is *not* like the SchemaSQL metavariable declarations, in that SchemaSQL allows complex interdependent declarations whereas FISQL declarations are independent of one another. The interdependency of SchemaSQL declarations significantly complicates its semantics and is unintuitive for SQL users, as declarations in SQL are independent. Furthermore, SchemaSQL column variables are superfluous and not included in FISQL.

Another crucial difference between FISQL and SchemaSQL is that FISQL uses **on** and/or **into** keywords for dynamic schemas. In contrast, SchemaSQL uses an extension of the “**create view**” construct for this. In SchemaSQL, a query in the body of a **create view** may change semantics [12]. This means SchemaSQL queries are harder to understand and formalize, and dynamic schemas may not appear in subqueries. Furthermore, SchemaSQL relies on a non-deterministic *merge* operation [12]. Although special cases of *merge* can be deterministic [3, 10, 21], in general *merge* is expensive [21].

Semantically, an important difference between FISQL and SchemaSQL is that FISQL has an equivalent extended relational algebra, FIRA [20]. Attempts have been made to underpin fragments of SchemaSQL with relational versions of tabular algebra operators [3, 11], such as *fold* and *unfold* [8]. A major limitation of this approach is that the original tabular algebra utilizes a richer data model, so semantic information and complex notions of validity come into play in the SchemaSQL transformation [3, 10, 11, 20]. For example, a *fold* can only be applied to the result of a previous *unfold*, and the columns involved in the *unfold* must have been stored as extra (semantic) information. Nonetheless, some optimization rules can be stated using the restricted tabular algebra operators. These can readily be seen to be special cases of some of the rules developed in Section 3.

These differences explain why reported results of implementations of SchemaSQL are limited to fixed schema, single relation cases, or else procedural simulations of a single, limited, flat dynamic schema operation [11]. In contrast, we have developed a fully multidatabase, dynamic schema implementation of FISQL including support for fully compositional, arbitrarily nested subqueries.

There is some work on procedurally unwinding SchemaSQL queries as an implementation strategy [1]. However, the implementation in [1] only investigated the possibility of a single resulting **union** query, and is thus restricted to a special case of our federate operator, Φ .

Our work is related to recent implementations of PIVOT and UNPIVOT [2, 21]. The implementation of dynamic schemas in [2] includes aggregation for collapsing conflicting values in a τ followed by a *merge*. Recent, related work shows how PIVOT and UNPIVOT can be generalized in an extended version of FIRA including a deterministic optimal tuple merge operation [21]. Future work will extend our engine to include aggregation.

Recent work applies FIRA transformations to delineate a search space for semi-automatically discovering integration mappings [5]. Our engine can be utilized to execute discovered mappings, since these take the form of FIRA queries.

6. CONCLUSION AND FUTURE WORK

In this paper, we have reported on an implementation of FISQL/FIRA for transformationally complete querying in multidatabase systems. Our implementation includes a full FISQL parser and translator, as well as a FIRA evaluation engine, and supports heuristic rule-based optimization in the style of RA optimization. We support dynamic schemas, null values, and heterogeneous atomic types. Optimization methods such as automatic syntactic removal of metavariables, generalized algebraic FIRA plan transformations, and the employment of source filter queries clearly distinguish us from previous, ad hoc approaches involving only metavariable unwinding. Our experiments show that

the approach is feasible for TPC type data having dynamic schemas and residing in multiple sources. We are the first to implement a relational query processor and optimizer for dynamic schemas on this scale.

Further interesting projects include adding aggregation to FISQL; studying the impact of network latency on FISQL query processing; as well as augmenting the engine with support for heterogeneous data models, such as semi-structured or nested. In general, the semantics provided by FIRA yields a solid basis for future expansion of our engine.

7. ACKNOWLEDGMENTS

Many thanks to members of the IU Database Lab for comments on previous drafts and presentations of this paper, especially Edward Robertson and George Fletcher. Also, thanks to Lucian Popa and Tom Hynes for reading a previous draft and giving comments. Finally, we thank the anonymous reviewers for their insightful comments.

8. REFERENCES

- [1] Francois Barbancon and Daniel P. Miranker. *Implementing Federated Database Systems by Compiling SchemaSQL*. IDEAS 2002.
- [2] Conor Cunningham, César A. Galindo-Legaria, and Goetz Graefe. *PIVOT and UNPIVOT: Optimization and Execution Strategies in an RDBMS*. VLDB 2004.
- [3] Keir B. Davis and Fereidoon Sadri. *Optimization of SchemaSQL Queries*. IDEAS 2001.
- [4] R. Dhamanka, Y. Lee, A. Doan, A. Halevy, and P. Domingos. *iMAP: Discovering Complex Semantic Matches Between Database Schema*. SIGMOD 2004.
- [5] George H. L. Fletcher and Catharine M. Wyss. *Data Mapping as Search*. EDBT 2006.
- [6] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, and J. Widom. *The TSIMMIS approach to mediation: Data models and Languages*. Journal of Intelligent Information Systems (JIIS), March/April 1997. Vol. 8, no. 2, pp 117-132.
- [7] John Grant, Witold Litwin, Nick Roussopoulos, and Timos Sellis. *Query Languages for Relational Multidatabases*. VLDB Journal, vol.2. pp 153-171, 1993
- [8] Marc Gyssens, Laks V.S. Lakshmanan, and Iyer N. Subramanian. *Tables as a Paradigm for Querying and Restructuring*. PODS96, pp. 93-103.
- [9] Laura Haas, Renee Miller, Bartholomew Niswonger, Mary Tork Roth, Peter Schwarz, and Edward Wimmers. *Transforming Heterogeneous Data with Database Middleware: Beyond Integration*. IEEE Bulletin of the Technical Committee on Data Engineering, March 1999. Vol. 22, no. 1, pp 31-36.
- [10] Qi He and Tok Wang Ling. *Extending and Inferring Functional Dependencies in Schema Transformation*. CIKM 2005.
- [11] L.V.S. Lakshmanan, F. Sadri, and S.N. Subramanian. *On Efficiently Implementing SchemaSQL on an SQL Database System*. VLDB 1999.
- [12] L.V.S. Lakshmanan, F. Sadri, and S.N. Subramanian. *SchemaSQL – an extension to SQL for multidatabase interoperability*. TODS Vol. 26, No. 4, December 2001.
- [13] Maurizio Lenzerini. *Data Integration: A Theoretical Perspective*. PODS 2002: 233-246.
- [14] Lin Liu. *SIGMOD Record: Special Issue on Semantic Integration*. 33:4, December 2004.
- [15] Witold Litwin, M. Ketabchi, and Ravi Krishnamurthy. *First Order Normal Form for Relational Databases and Multidatabases*. SIGMOD RECORD, Vol. 20, No. 4, December 1991.
- [16] Peter McBrien and Alexandra Poulouvasilis. *Data Integration by Bi-Directional Schema Transformation Rules*. ICDE 2003.
- [17] Renée J. Miller, Mauricio A. Hernández, Laura M. Haas, Lingling Yan, C.T. Howard Ho, Ronald Fagin, and Lucian Popa. *The Clio Project: Managing Heterogeneity*. SIGMOD Record, 30:1, March 2001.
- [18] Amit P. Sheth and James A. Larson. *Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases*. ACM Computing Surveys, 22:3, 1990.
- [19] M. T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems*, 2nd ed. Prentice Hall, 1999.
- [20] Catharine M. Wyss and Edward L. Robertson. *Relational Languages for Metadata Integration*. TODS, Vol. 30, No. 2, June 2005.
- [21] Catharine M. Wyss and Edward L. Robertson. *A formal characterization of PIVOT/UNPIVOT*. CIKM 2005: 602-608.
- [22] <http://www.microsoft.com/net>,
<http://www.microsoft.com/vcsharp>
- [23] <http://wwwantlr.org>
- [24] <http://www.microsoft.com/sql>

APPENDIX

A. EBNF FOR FISQL

<code><query></code>	::=	<code><query_term> [(UNION MINUS) <query>]</code>
<code><query_term></code>	::=	<code><query_specifier></code> <code>(<query>)</code>
<code><query_specifier></code>	::=	<code>SELECT [DISTINCT] <column_list></code> <code>INTO <column_expr></code> <code>[FROM <variable_decl> {,<variable_decl>}*</code> <code>[WHERE <condition>]]</code>
<code><column_list></code>	::=	<code><column_decl> {,<column_decl>}*</code> <code>* [DROP <drop_colref> {,<drop_colref>}*]</code>
<code><drop_colref></code>	::=	<code><identifier> [.<identifier>]</code>
<code><column_decl></code>	::=	<code><column_expr> [AS <literal>]</code> <code><column_expr> ON <column_expr></code>
<code><variable_decl></code>	::=	<code><source_db> [(meta_decl) [(meta_decl)]]</code> <code>[AS] <identifier></code>
<code><source_db></code>	::=	<code>[:]<identifier></code> <code>(<query>)</code>
<code><meta_decl></code>	::=	<code>: <identifier></code> <code>. <identifier></code>
<code><column_expr></code>	::=	<code><literal></code> <code><identifier> [.<identifier> [.<identifier>]]</code>
<code><condition></code>	::=	<code><and_cond> [OR <condition>]</code>
<code><and_cond></code>	::=	<code><unary_cond> [AND <and_cond>]</code>
<code><unary_cond></code>	::=	<code><comp_expr></code> <code>NOT <unary_cond></code> <code>(<condition>)</code>
<code><comp_expr></code>	::=	<code><column_expr> <comp_op> <column_expr></code>
<code><comp_op></code>	::=	<code>= != <> <= < > >=</code>
<code><literal></code>	::=	<code><integer_literal></code> <code><string_literal></code> <code><regular_ident></code> <code><delimited_ident></code>
<code><integer_literal></code>	::=	<code>[- +]{<digit>}+</code>
<code><string_literal></code>	::=	<code>'{~}'* , '{' {~}'* , }*</code>
<code><regular_ident></code>	::=	<code>((<letter>) _ @ #) {(<letter>) (<digit>) _ @ # \$}*</code>
<code><delimited_ident></code>	::=	<code>"{~}"* " "{~}"* " "*</code> <code>[{~}]*]</code>
<code><letter></code>	::=	<code>a..z A..Z</code>
<code><digit></code>	::=	<code>0..9</code>

Figure 13: EBNF grammar for FISQL as implemented.