# eXtensible Relational Databases: a relational approach to interoperability

## ABSTRACT

Successful data integration relies on knowledge of (i) the structural shape of constituent data sources, and (ii) semantic knowledge of how structural terms map together, or how structure maps to structure. In this paper, we propose a simple extension to the relational model which allows these types of metaknowledge to be stored directly within relational sources and then incorporated into queries over these sources. This extension is termed the eXtensible Relational Database (XRDB) due both to affinities of the concept of metaknowledge with semantic web ideology, as well as due to use of the framework in storing active XML metadata – semantic metadata with functionality – in relational databases. While XRDB entails storing data from two fundamentally different conceptual levels within the relational model, the query language we propose is a simple extension of ordinary SQL and allows for seamless usage of both data and metaknowledge. A crucial advantage of the extended language is that it allows dynamically generated mediated views to be computed from metaknowledge stored in the data sources. These views address problems ranging from semantic heterogeneity to structural dissimilarity, and even allow us to quantify dissimilarity of data sources. A prototype implementation of our framework exists, and we briefly describe its underlying architecture. We conclude the paper by discussing how our query language compares to existing metadata integration languages.

## Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design—*Data models*; H.2.3 [**Database Management**]: Languages—*Query languages*; H.2.5 [**Database Management**]: Heterogeneous Databases—*Data translation*

## General Terms

Languages, Theory

## Keywords

Database integration, interoperability, metadata, query languages, relational algebra, dynamic mediated schema

## 1. INTRODUCTION

Current tools to support data sharing are insufficient. In this paper, a generalization to the relational database model with intrinsic support for metadata is proposed as a framework for constructing a multi-database (or federated database) out of relational data sources. The following theses underly the research described here.

1. We adopt the premise that having a rich set of metadata is the key to effective data sharing and to achieving interoperability. This premise agrees with the prevailing view in the database community, and is a primary motivation in the efforts behind the development of XML.[1] However, just as there are many advantages to representing XML data (i.e., document) and metadata (i.e., schema) within the same framework — the semi-structured data model, similar advantages can be gained if metadata is structured within the same model as the relational data they describe.

   THESIS 1: *Structural and semantic metadata for relational data sources can be effectively and naturally represented within the relational model.*

   Note that this is different from allowing data to be interpreted as metadata and vice versa – the status of metadata and data are clearly delineated. In Section 6, we will consider this point in greater details.

2. The most common approach to data sharing among heterogeneous data sources is to develop a *mediated schema* [8]. However, the difficulty in the creation and the evolution of such schema is the cause for most heterogeneous database system being unscalable [10, 2]. The foundational work on SchemaSQL [5, 6] is the first to attempt to develop a relational multi-database system that supports query-driven, dynamically generated mediated schema while staying within the relational framework. Our work adopts the same view.

   THESIS 2: *Scalable solutions to data integration should support dynamic generation of mediated schemas within the query language of the data model.*

We elaborate on each of the points briefly.

*Metadata Representation:.* From a data management perspective, one reason for the excitement over XML is its ability to annotate data with metadata in an uniform manner. While XML and numerous other conceptual languages exist for conveying metadata (such as Entity Relationship Diagrams and the Unified Modeling

[1]Indeed, metadata forms the core in many other database related endeavors such as the semantic web and automatic database design (e.g. [15]).

Language), what has been overlooked is the potential for extending the relational model itself with support for metadata integration. Certainly, the semantic simplicity of the relational data model is what has made the relational database industry an unprecedented success. If it turns out that the model can be used to capture most (or perhaps all) of the meta-level information necessary to facilitate interoperability, then we gain conceptually by being able to build relational databases that are self-describing, and practically by being able to use existing techniques and tools of relational database systems to manage metadata. Moreover, we avoid the additional complexity that almost always comes with combining the relational data model with a different data model.

*Dynamic Mediated Schema.* Knowledge sharing and database interoperability solutions are currently performed on an *ad hoc* basis and mostly through human intervention, *even if the semantics of the data is compatible and only the underlying structure differs*. In a federation comprised of relational data sources, for example, the dominant architecture remains *wrapper-based* and involves the creation of special-purpose wrappers for translating global data manipulation into locally manifested operations [7]. Central to the translation is in the creation of a mediated schema – a schema that reconciles all relevant relations, attributes, and constraints of interest to the federation from the interoperating data sources. Aside from being time-consuming to produce, mediated schemas thus created are not easily adaptable to changes in the underlying sources, nor can they be easily extended to incorporate new data sources. A more attractive approach, then, is to allow for the dynamic generation of mediated schemas based on source-specific (and ideally source-supplied) metadata in the query language.

*XRDB overview:.* In this paper we consider the eXtensible Relational Database (XRDB) to address the two points discussed above. XRDB is a conceptual extension to the relational database model that integrates metadata within the same model. At the core of XRDB are two sets of relations: meta-relations and object-relations. Data in the meta-relation represent design-specific information *about* the object-relations.

Based on the relational representation of metadata, we introduce a simple yet powerful language extension to SQL to facilitate query-driven dynamic mediated schema creation. The extension relies on the basic observation that, when applied to metadata consisting of relation and attribute names, the result of a query are tuples that represent a set of *virtual mediated schemas* from which source-specific queries can be constructed.

We acknowledge that in practice, most existing relational database systems already maintain certain metadata explicitly — typically via a system catalog. However, the metadata in these systems are not part of the conceptual framework, and are therefore not designed to be used in the ways envisioned in this paper. In particular, users do not have the freedom to specify which meta-relation particular metadata should reside, nor do they have the ability to extend meta-relations with additional attributes.

The organization of the paper is as follows. In Section 2, the basics of meta-relations and XRDB are introduced. Sections 3 and 4 discuss extensions to relational query languages and SQL that take advantage of the presence of metadata in XRDB. Section 5 illustrates the power of the XRDB framework in an interoperability scenario. Connections to related work are examined briefly in Section 6, and the design of an on-going implementation effort is discussed in Section 7.

## 2. META-RELATIONS AND XRDB

We assume countably infinite sets of *object-attributes* $\mathcal{A}$ and *object-values* $\mathcal{V}$. As usual, we assume that $\mathcal{A} \cap \mathcal{V} = \emptyset$.

Each attribute $a$ has an associated set of values $\Delta(a) \subseteq \mathcal{V}$. Given a subset $X$ of $\mathcal{A}$, an $X$-tuple $t$ is a mapping from $X$ into $\mathcal{V}$ such that each $a \in X$ is mapped to an element of $\Delta(a)$.

An *object-relation schema* $R$ is a finite subset $\{a_1, ..., a_k\}$ of $\mathcal{A}$, and an *object-relation* is a finite set of $R$-tuples. We will often denote an $R$-tuple $t$ using the notation $\langle a_1 : x_1, ..., a_k : x_k \rangle$, where $x_i = t[a_i]$. Each object-relation schema and attribute of a schema has an associated name (identifier). The set of all object-relation schema names and the set of all object-attribute names are denoted $\Omega$ and $\Xi$, respectively.

A set of object-relation schemas $D$ is called an *object-database schema*, and an *object-database* is a set of object-relations, one object-relation over each object-relation schema in $D$.

Next, we assume countably infinite sets of *meta-attributes* $\mathcal{A}_m$ and *meta-values* $\mathcal{V}_m$, $\mathcal{A}_m \cap \mathcal{V}_m = \emptyset$, $\mathcal{A}_m \cap \mathcal{A} = \emptyset$, and $(\Omega \cup \Xi) \subseteq \mathcal{V}_m$. The notions of *meta-relation schema* and *meta-relation* are defined analogously to object-relation schema and object-relation, respectively. Given an attribute $a$ of some meta-relation schema, we say $a$ is an *$\Omega$-attribute* if $\Delta(a) = \Omega$, and *$\Xi$-attribute* if $\Delta(a) = \Xi$.

A *meta-database schema* $D_m$ is a set of meta-relation schemas, and a *meta-database* is a set of meta-relations, one meta-relation over each meta-relation schema in $D_m$.

**Notation.** Given a (meta- or object-) relation $R$, $\phi(R)$ denotes the schema of $R$.

**Definition (XRDB).** An XRDB-schema is (MDB,ODB,$\alpha$,$\mathcal{D}$) where MDB is a set of meta-relation schemas, ODB is a set of object-relation schemas, $\alpha$ is a function that maps values in $\Omega$ to the corresponding relation schemas in ODB, and for each object-relation schema $R \in$ ODB, there is a function $R.\alpha \in \mathcal{D}$ that maps names of $\phi(R)$ to the corresponding $R$-attributes.

An XRDB consists of a set of meta-relations over MDB and a set of object-relations over ODB.

**Remark.** Typically, what distinguishes a meta-relation from a object-relation is that it contains either object-relation or object-attribute names. This is not a requirement, however, as examples below will illustrate. Also, a number of other types of meta-level information exist about object-databases, including, for example, key and foreign key constraints. In this paper, in particular with respect to the development of the query language, we focus attention on relation and attribute names. Additional meta-datatypes are discussed in Section 7 and will be explored more fully in future research.

**Notation.** In examples, we prefix meta-relation names and their associated attributes with the letter `m`. Moreover, we suffix the names of $\Omega$-attributes with `Rel`, and names of $\Xi$-attributes with `Att`.

Obvious examples of meta-relations include those for representing structural information about object relations. Schemas of two such meta-relations are shown below. The first is to specify the attributes associated with a given object-relations, and the second to represent foreign keys among object-relations.

1. We call this meta-relation `mAttribute`. Its schema is

```
mAttribute(
  /* name of an object-relation */
  mRel,
  /* name of an attribute in
     the relation              */
  mAtt
)
```

The tuple `<mRel:manager, mAtt:SSN>` represents that `SSN` is an attribute in $\phi($`manager`$)$. The usual notation is `manager.SSN`.

2. We call this meta-relation `mForeign_key`. Its schema is

```
mForeign_key(
  /* the referencing relation */
  mSourceRel,
  /* the referencing attribute
     in mSourceRel            */
  mSourceAtt,
  /* the referenced relation */
  mTargetRel,
  /* the referenced attribute
     in mTargetRel           */
  mTargetAtt,
)
```

An `mForeign_key`-tuple

```
< mSourceRel:employee, mSourceAtt:mgrId,
  mTargetRel:manager, mTargetAtt:id >
```

indicates that `employee.mgrId` has a foreign key constraint to `manager.id`.

Other useful meta-relations may be, for example, to capture database design information. We give two examples below.

1. Depending on applications, we may wish to maintain information about whether a given object-relation is derived from an entity or a relationship in an ER design diagram. Suppose the ODB contains two relations, one representing employee and the second representing a "works for" relationship. This information may be captured in the meta-relation below.

```
{ <mRelKeyword:employee,
       mERD_type:entity>,
  <mRelKeyword:works_for,
       mERD_type:relationship>
}
```

By our naming convention, one can see that the schema for the above relation contains neither $\Omega$ nor $\Xi$ attributes. Intuitively, the idea is that `employee` and `works_for` may represent semantic values [13] associated with object-relations that *mean* employee (e.g., staff, professor, etc.) and "works for", respectively. A more detailed discussion of using keywords to resolve semantic heterogeneity appears in Section 5.

2. A meta-relation may be used to capture syntactic restrictions on attributes of object-relations. The meta-relation below, for instance, indicates the syntactic patterns of attributes related to social-security numbers and phone numbers.

```
{ <mAttKeyword:SSN,
     mPattern:"___-__-____">,
  <mAttKeyword:Phone,
     mPattern:"____-___-____" >}
```

The observation that relations can be used to manage the metadata of relational databases is by no means new. Most relational database management systems provide metadata support to various extents, and numerous research projects into database interoperability issues rely on the self-describing nature of the relational model (e.g., [4, 2]). To facilitate transparent object-data querying based on queries over metadata, we propose an extension to SQL of the form

```
[UNION|INTERSECT] <expression1>
WITH <parameters>
BASED ON <expression2>
```

where `<expression2>` is an ordinary SQL query over meta-relations, `<expression1>` is a parameterized SQL query whose formal parameters are specified in `<parameters>`, and they take on appropriate values that result from `<expression2>`. In the next section, we introduce the XRDB Template Query Language (XTQL) to formalize the semantics for the above SQL extension.

## 3. XTQL: A TEMPLATE QUERY LANGUAGE FOR XRDB

With metadata represented in the same fashion as object data, queries *about* the object database may be posed via ordinary relational algebra expressions.

**Definition (Meta-Query).** Let $D_m = \{R_1, ..., R_n\}$ be a meta-database schema, then any relational algebra expression over $D_m$ is a *meta-query* over $D_m$. The semantics of a meta-query is given via the primitive relational operations [3].

A simple example of a meta-query is the following. We abbreviate meta-relations `mForeign_key` and `mAttribute` defined in the last section by $fk$ and $at$, respectively. To find all object-relations that have an attribute called `SSN` and the relation is referenced by the object-relation `employee`:

$$\pi_{fk.\texttt{mTargetRel}}(\sigma_{fk.\texttt{mSourceRel}=\texttt{"employee"}\wedge fk.\texttt{mSourceRel}=at.\texttt{mRel}\wedge at.\texttt{mAtt}=\texttt{"SSN"}}(fk \times at))$$

**Example.** Consider the example XRDB in Figure 1. The database contains one meta-relation called `mKeywords` and two object relations: `student` and `professor`. As before, one may think of the value `SSN` in the `mKeyword` attribute of `mKeywords` as an annotation by the database designer to indicate that semantically, `student.sid` and `professor.pid` are the same as the social security number.

The meta-query

$$\pi_{\texttt{mRel,mAtt}}(\sigma_{\texttt{mKeyword = "SSN"}}(\texttt{mKeywords})) \quad (1)$$

returns the meta-relation

```
{ <mRel:student, mAtt:sid>,
  <mRel:professor, mAtt:pid> }
```

Metadata embody the knowledge necessary to structure queries over object-data, and conceptually, any traditional *wrapper-based* approach to manipulating object-data from one source into data for another is performing the transformation based on this knowledge. However, the knowledge of the metadata is hard-coded into the logic of the program (or queries [16]). With metadata explicitly represented in XRDB, it is now possible to define *generic* mappings from meta-relations derived through meta-queries to object-relations. Abstractly, we call these mappings T-functions. Ideally, the language for specifying T-functions should adhere closely to relational algebra/SQL. A T-function for the above example we will call `mergeSSN` is to map the meta-relation mKeywords

| mRel | mAtt | mKeyword |
|---|---|---|
| student | sid | SSN |
| student | name | name |
| professor | pid | SSN |
| professor | name | name |
| professor | rank | designation |

student

| sid | name |
|---|---|
| 123 | joe |
| 234 | mary |

professor

| pid | name | rank |
|---|---|---|
| 111 | joe | associate |
| 222 | pam | assistant |

**Figure 1: An Example XRDB**

```
{ <mRel:student,mAtt:sid>,
  <mRel:professor,mAtt:pid> }
```

to the object-relation

```
{ <ssn:123>,<ssn:234>,
  <ssn:111>,<ssn:222> }
```

Intuitively, this T-function "integrates" the `sid` and `pid` attributes of `student` and `professor`, respectively.

The remainder of the section is devoted to formulating the syntax and semantics of a class of T-functions within XTQL.

**Definition.** Suppose $a$ is an $\Omega$-attribute in a meta-relation schema, the expression $\$a$ is called a *relation term*. Suppose $b$ is an $\Xi$-attribute, then $\$b$ is called an *attribute term*.

**Definition (T-expression and Namespace).** A T-expression $\mathcal{X}$ and its associated namespace, $\Psi(\mathcal{X})$, over a meta-relation schema $R(a_1, ..., a_k)$ are defined as follows.

1. A relation term $\$a$ is a T-expression. Its associated namespace is $\{\$a\}$.
2. Suppose $\mathcal{X}$ is a T-expression and $r$ is a name that does not occur in $\Psi(\mathcal{X})$. Then $\rho_r(\mathcal{X})$ is a T-expression and $\Psi(\rho_r(\mathcal{X})) = \{r\}$.
3. Suppose $\mathcal{X}_1, \mathcal{X}_2$ are T-expressions. Then $\mathcal{X}_1 \times \mathcal{X}_2$ is a T-expression and $\Psi(\mathcal{X}_1 \times \mathcal{X}_2) = \Psi(\mathcal{X}_1) \cup \Psi(\mathcal{X}_2)$.
4. Suppose $\mathcal{X}$ is a T-expression, $\$b_1, ..., \$b_m$ are attribute terms, and $n_1, ..., n_m \in \Psi(\mathcal{X})$. Then $\pi_{n_1.\$b_1,...,n_m.\$b_m}(\mathcal{X})$ is a T-expression, and $\Psi(\pi_{n_1.\$b_1,...,n_m.\$b_m}(\mathcal{X})) = \Psi(\mathcal{X})$.
5. Suppose $\mathcal{X}$ is a T-expression, $a$ is an object-attribute, $\$b$ is an attribute term, $n \in \Psi(\mathcal{X})$, and $c$ is a literal (e.g., string constant, number, etc.). Then t-boolean expressions over $\mathcal{X}$ are those expressions generated from the following grammar

| | | |
|---|---|---|
| t-boolean | $\rightarrow$ | t-comp $\{\wedge \mid \vee\}$ t-comp |
| t-comp | $\rightarrow$ | t-term op t-term |
| op | $\rightarrow$ | $= \mid < \mid > \mid \leq \mid \geq \mid \neq$ |
| t-term | $\rightarrow$ | $n.\$b \mid a \mid c$ |

   Given a t-boolean expression $e$ over a T-expression $\mathcal{X}$, the expression $\sigma_e(\mathcal{X})$ is a T-expression, and $\Psi(\sigma_e(\mathcal{X})) = \Psi(\mathcal{X})$.

**Example.** Consider the schema $\{\texttt{mRel}, \texttt{mAtt}\}$ associated with the output of meta-query (1). Two example T-expressions over this schema are

$$\begin{aligned} X_1 &= \pi_{r.\$\texttt{mAtt}}(\rho_r(\$\texttt{mRel})) \\ X_2 &= \sigma_{r.\$\texttt{mAtt}=\texttt{"joe"}}(\rho_r(\$\texttt{mRel})) \end{aligned}$$

T-expressions form the basis for specifying T-functions, and their semantics are given by associating an object-relation with each tuple of a meta-relation. In the next two definitions, we assume $R$ is a meta-relation, and $\mathcal{X}$ is a T-expression defined over the schema $\phi(R) = \{a_1, ..., a_k\}$. Recall that $\alpha$ of an XRDB schema is a function that maps object-relation names to object-relations, and for each object-relation $r$, $r.\alpha$ is a function that maps each attribute name to an attribute in $r$.

**Definition.** Given a tuple $t$ and a T-expression $\mathcal{X}$, the *evaluation of* $\mathcal{X}$ *over* $t$, $\mathcal{X}(t)$, is the object-relation obtained by first substituting $\alpha(t[a])$ for each relation term $\$a$ in $\mathcal{X}$, $r.\alpha(t[b])$ for each attribute term $r.\$b$ in $\mathcal{X}$, and then evaluating the resulting ordinary relational algebraic expression if the usual schema restrictions are satisfied [3].

**Example.** Take the schema $\{\texttt{mRel}, \texttt{mAtt}\}$ again and consider the tuple $t$

```
< mRel:student, mAtt:name >
```

Using the relation `student` in Figure 1, the evaluation of the T-expressions $X_1$ and $X_2$ over $t$ from the previous examples are as follows. For $X_1$:

$$\begin{aligned} &\pi_{r.\$\texttt{mAtt}}(\rho_r(\$\texttt{mRel}))(t) \\ &\quad = \pi_{r.\alpha(t[\texttt{mAtt}])}(\rho_r(\alpha(t[\texttt{mRel}]))) \\ &\quad = \pi_{r.\texttt{name}}(\rho_r(\texttt{student})) \\ &\quad = \{\langle \texttt{name:joe} \rangle, \langle \texttt{name:mary} \rangle\} \end{aligned}$$

For $X_2$:

$$\begin{aligned} &\sigma_{r.\$\texttt{mAtt}=\texttt{"joe"}}(\rho_r(\$\texttt{mRel}))(t) \\ &\quad = \sigma_{r.\alpha(t[\texttt{mAtt}])=\texttt{"joe"}}(\rho_r(\alpha(t[\texttt{mRel}]))) \\ &\quad = \sigma_{r.\texttt{name}=\texttt{"joe"}}(\rho_r(\texttt{student})) \\ &\quad = \{\langle \texttt{name:joe}, \texttt{sid:123} \rangle\} \end{aligned}$$

Note that as T-expressions are constructed via relational algebra syntax, it would be desirable to enforce the usual schema restrictions of relational algebra in the definition of T-expressions. However, a T-expression is only *interpreted* as a relational algebra expression upon evaluation. It follows that a "compile-time" schema restriction is not possible. Instead, any schema requirement can only be enforced at the time of an evaluation, and if an attempt to reference a non-existent attribute of an object-relation is made, for instance, in projection, then a "run-time" error would result.

The evaluation of a T-expression over a meta-relation is the set obtained via a tuple-by-tuple evaluation.

**Definition.** The *evaluation of* $\mathcal{X}$ *over* a meta-relation $M$, $\mathcal{X}(M)$, is the set $\{T | T = \mathcal{X}(t),\ \text{for}\ t \in M\}$.

**Example.** Consider the result $R$ of meta-query (1):

```
{ <mRel:student,mAtt:sid>,
  <mRel:professor,mAtt:pid> }
```

Suppose $t_1$ and $t_2$ denote the two tuples in $R$, respectively. Then the evaluations $X_1$ and $X_2$ from the previous example are shown below.

$$\begin{aligned} &X_1(R) = \pi_{r.\$\texttt{mAtt}}(\rho_r(\$\texttt{mRel}))(R) \\ &\quad = \{\pi_{r.\$\texttt{mAtt}}(\rho_r(\$\texttt{mRel}))(t_1), \\ &\qquad \pi_{r.\$\texttt{mAtt}}(\rho_r(\$\texttt{mRel}))(t_2)\} \\ &\quad = \{\{\langle \texttt{sid:123} \rangle, \langle \texttt{sid:234} \rangle\}, \\ &\qquad \{\langle \texttt{pid:111} \rangle, \langle \texttt{pid:222} \rangle\}\} \end{aligned}$$

$$\begin{aligned}X_2(R) &= \sigma_{r.\texttt{\$mAtt}=\texttt{"joe"}}(\rho_r(\texttt{\$mRel}))(R)\\ &= \{\sigma_{r.\texttt{\$mAtt}=\texttt{"joe"}}(\rho_r(\texttt{\$mRel}))(t_1),\\ &\quad \sigma_{r.\texttt{\$mAtt}=\texttt{"joe"}}(\rho_r(\texttt{\$mRel}))(t_2)\}\\ &= \{\{\langle \texttt{name:joe}, \texttt{sid:123}\rangle\},\\ &\quad \{\langle \texttt{name:joe}, \texttt{pid:111}, \texttt{rank:associate}\rangle\}\}\end{aligned}$$

Intuitively, meta-queries allow metadata associated with multiple data sources to be structured into a single meta-relation. In turn, the set of object-relations returned from applying a T-expression to that meta-relation represents data collected from the data sources. Once obtained, the relations may be combine into a single relation by taking their union or intersection.

Next we define XTQL queries and their evaluations.

**Definition (XTQL Query).** Suppose $M$ is an XRDB over the schema (MDB,ODB,$\alpha$,$\mathcal{D}$).

META-QUERY A meta-query over MDB is an XTQL-query. As described previously, the evaluation of the query is based on standard relational algebra operations [3].

T-QUERY Suppose $Q$ is a meta-query over MDB and $\mathcal{X}$ is a T-expression over $\phi(Q)$. Then $\cup\mathcal{X}(Q)$ and $\cap\mathcal{X}(Q)$ are XTQL-queries.

We say the queries are *well-formed* if the schemas of the object-relations in the evaluation of $\mathcal{X}$ over $Q$ are pairwise union-compatible. For these queries, the evaluation of $\cup\mathcal{X}(Q)$ and $\cap\mathcal{X}(Q)$ are obtained by taking the union and intersection, respectively, of the set of relations in $\mathcal{X}(Q)$.

OBJECT-QUERY Suppose $T$ is well-formed T-query. Then any relational algebra expression over $\{\phi(T)\}\cup$ODB is an XTQL-query, and its evaluation is based on ordinary relational algebra operations.

**Example.** A query to find the SSN of all students and professors in the database of Figure 1 is the following.

$$\rho_{R(\texttt{ssn})}(\cup(\pi_{r.\texttt{\$mAtt}}(\rho_r(\texttt{\$mRel}))(Q))) \tag{2}$$

where $Q$ is the meta-query of Equation (1). The result of the query is

```
{ <ssn:123>,<ssn:234>,
  <ssn:111>,<ssn:222> }
```

Indeed, the query implements exactly the T-function `mergeSSN` discussed previously.

**Example.** Consider the meta-query $Q_3$ below over `mKeywords`.

$$Q_1 = \rho_{r_1}(\texttt{mKeywords}) \times \rho_{r_2}(\texttt{mKeywords})$$

$$Q_2 = \sigma_{r_1.\texttt{mAtt}=r_2.\texttt{mAtt}\wedge r_1.\texttt{mRel}\neq r_2.\texttt{mRel}}(Q_1)$$

$$Q_3 = \rho_{r(\texttt{m1},\texttt{m2},\texttt{mA})}(\pi_{r_1.\texttt{mRel},r_2.\texttt{mRel},r_2.\texttt{mAtt}}(Q_2))$$

$Q_3$ retrieves all pairs of object-relation names that have the same attribute name. In this case, the resulting meta-relation $R$ is

```
{<m1:student, m2:professor, mA:name>,
 <m1:professor, m2:student, mA:name>}
```

Then, XTQL-query $Q_4$ below retrieves all pairs of professors and students that have the same name:

$$X_3 = \sigma_{r_1.\texttt{\$mA}=r_2.\texttt{\$mA}}(\rho_{r_1}(\texttt{\$m1}) \times \rho_{r_2}(\texttt{\$m2}))$$

$$Q_4 = \rho_{r.(\texttt{a1},\texttt{a2},\texttt{a3},\texttt{a4},\texttt{a5})}(\cup X_3(Q_3)) =$$

```
{ <a1:123, a2:joe, a3:111,
      a4:joe, a5:associate>,
  <a1:111, a2:joe, a3:associate,
      a4:123, a5:joe> }
```

These examples provide a glimps of the power of XRDB and XTQL for dealing with heterogeneity in relational designs. In fact, query (2) will retrieve all SSN's in the database regardless of the number of relations that have semantically equivalent attributes. For example, the database may maintain separate relations for `staff` and `administrators`. As long as the appropriate metadata are specified in `mKeyword`, the query need not be modified. Thus, XRDB adheres closely to the advice that data, not code (which includes queries), is a better candidate for reuse [11].

**Proposition.** $XTQL \subseteq \text{LOGSPACE}$. □

Selections on the result of a T-query may be driven inward to reduce the size of the relations created for each instance of the T-query. First, we adopt the convention that the schema of the union and intersection of a set of relations is the schema of the first relation [1]. In particular, suppose $\mathcal{X}(Q) = \{R_1, ..., R_n\}$. Then $\phi(\cup\mathcal{X}(Q)) = \phi(\cap\mathcal{X}(Q)) = \phi(R_1)$.

**Proposition.** Suppose $\cup\mathcal{X}(Q)$ is a T-query with $\phi(\cup\mathcal{X}(Q)) = \{a_1, ..., a_k\}$. Suppose moreover, for each relation $R \in \mathcal{X}(Q)$, $\phi(R) = \{a_1^R, ..., a_k^R\}$.

Let $e$ be a boolean expression constructed via literals and attributes over $\phi(\cup\mathcal{X}(Q)) = \{a_1, ..., a_k\}$, and for each $R \in \mathcal{X}(Q)$, let $e_R$ denote the replacement of each attribute $a_i$ occurring in $e$ by $a_i^R$. Then

$$\sigma_e(\cup\mathcal{X}(Q)) = \cup\{\sigma_{e_R}(R) | R \in \mathcal{X}(Q)\}.$$

Similarly, for the T-query $\cap\mathcal{X}(Q)$, the following equality holds.

$$\sigma_e(\cap\mathcal{X}(Q)) = \cap\{\sigma_{e_R}(R) | R \in \mathcal{X}(Q)\}.$$

□

Technically, the above results are not query rewriting since the righthand sides of the equalities do not form an XTQL-query. However, regarded as transformations on internal tree representations of queries, the results are just as practically significant with respect to query optimization.

A more interesting class of equivalences is to drive selections that occur outside of a T-expression inside the expression. Such a transformation modifies functions denoted by T-expressions into different functions.

**Proposition.** Suppose $Q$ is a meta-query, $\mathcal{X}$ is a T-expression over $\phi(Q)$, and for each tuple $t \in Q$, $e_t$ is a boolean expression constructed via literals and attributes of the object-relation schema $\phi(\mathcal{X}(t))$.

We assume $e_t$ is of the form $c_1\Theta_1...\Theta_{n-1}c_n$, where each $\Theta_i$ is either $\vee$ or $\wedge$, and each $c_i$ is a *comparison* of the form $b_1^i$ op $b_2^i$, where op $\in \{<, >, \leq, \geq, =, \neq\}$. Then

1. If $\mathcal{X} = \sigma_e(\mathcal{X}_1)$, then
$$\sigma_{e_t}(\mathcal{X}(t)) = \sigma_{e\wedge e_t}(\mathcal{X}_1)(t).$$
2. If $\mathcal{X} = \pi_{n_1.\$b_1,...,n_m.\$b_m}(\mathcal{X}_1)$ and for each attribute $a$ that occurs in $e_t$, $a \in \phi(\mathcal{X}(t))$, then
$$\sigma_{e_t}(\mathcal{X}(t)) = (\pi_{n_1.\$b_1,...,n_m.\$b_m}(\sigma_{e_t}(\mathcal{X}_1))(t)).$$
3. Suppose $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2$. Let $t_1$ and $t_2$ denote $t$ restricted to the relation and attribute terms occurring in $\mathcal{X}_1$ and $\mathcal{X}_2$, respectively. Moreover, let $e_{t_1}$ and $e_{t_2}$ denote the boolean expressions obtained from $e_t$ by removing those comparisons

that contain attributes in $\phi(\mathcal{X}_2(t_2))$ and $\phi(\mathcal{X}_1(t_1))$, respectively.

Then, if $e_t$ does not contain a comparison $a$ op $b$ such that $a \in \phi(\mathcal{X}_1(t_1))$ and $b \in \phi(\mathcal{X}_2(t_2))$ or vice versa, then

$$\sigma_{e_t}(\mathcal{X}(t)) = (\sigma_{e_{t_1}}(\mathcal{X}_1) \times \sigma_{e_{t_2}}(\mathcal{X}_2))(t).$$

□

**Example.** We illustrate the last case of the proposition. Consider the database of Figure 1, the tuple $t$

```
<mRel1:student, mRel2:professor>
```

of a meta-relation with schema `{mRel1,mRel2}`, and the T-expression

$$X = (\rho_{r_1}(\$\texttt{mRel1}) \times \rho_{r_2}(\$\texttt{mRel2}))$$

Let $e_t$ denote the boolean expression $\texttt{sid} < 120 \wedge \texttt{pid} < 120$. We have

- $t_1 = $ `<mRel1:student>`
- $t_2 = $ `<mRel2:professor>`
- $\phi(\rho_{r_1}(\$\texttt{mRel1})(t_1)) = \{\texttt{sid,name}\}$
- $\phi(\rho_{r_2}(\$\texttt{mRel2})(t_2)) = \{\texttt{pid,name,rank}\}$

Then the proposition states that

$$\sigma_{e_t}(X)(t) = (\sigma_{e_{t_1}}(\rho_{r_1}(\$\texttt{mRel1})) \times \sigma_{e_{t_2}}(\rho_{r_2}(\$\texttt{mRel2})))(t)$$

where $e_{t_1} = $ `sid < 120` and $e_{t_2} = $ `pid < 120`.

## 4. AN EXTENSION TO SQL

The semantics of the SQL extension described prior to Section 3 is now clear. As meta-queries can be formulated in ordinary SQL, the necessary key extension is to mimic the idea of T-queries. To make the interaction between the T-expression and the meta-query clearer, we introduce the *meta-attribute list* which is an expression of the following form:

```
[a1 <a1-renamed>, ..., an <an-rename>]
```

The idea is each `ai` is an attribute of meta-relation schema from a given meta-query, and that `<ai-renamed>` is the corresponding name used inside the T-expression.

As shown previously, the general syntax of our SQL extension is as follows.

```
[UNION|INTERSECT] <expression1>
WITH <parameters>
BASED ON <expression2>
```

Specifically, `<expression1>` is a T-expression, `<parameters>` is a meta-attribute list, and `<expression2>` is a meta-query. The following example illustrates how XTQL-Query (2) can be written in the above syntax.

```
UNION (SELECT $a ssn              (I)
       FROM $r)
WITH [K.mRel r, K.mAtt a]
BASED ON (SELECT K.mRel, K.mAtt   (II)
          FROM   mKeywords K
          WHERE  K.mKeyword = "SSN")
```

Subquery (II) finds, from `mKeywords`, all those relations with an attribute that is semantically equivalent to `"SSN"`. Each qualifying tuple becomes a tuple in the meta-relation with the schema

```
[K.mRel r, K.mAtt a]
```

In turn, `$r` and `$a` are substituted and evaluated to its named object in subquery (I). The results of each instance of subquery (I) is unioned and returned as the result of the overall query.

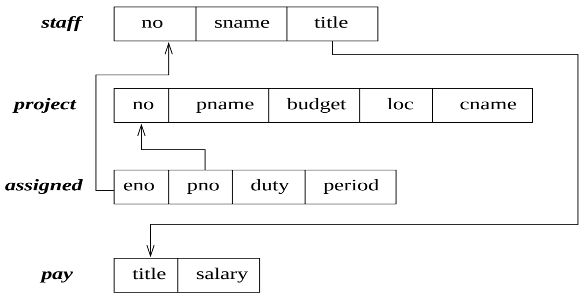

**Figure 2: DB$_1$: A Staff Database**

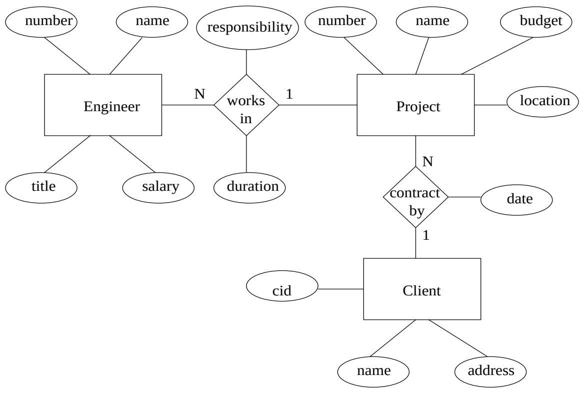

**Figure 3: DB$_2$: An Engineer Database**

## 5. INTEROPERABILITY

We use the following scenario for our discussion. Assume two database designs (adapted from [8]). The first, shown in Figure 2, is expressed as a set of relational schemas, while the second, shown in Figure 3, is represented as an Entity-Relationship Diagram.

Much of the structural aspects of the metadata associated with both designs can be extracted automatically. In the case of DB$_2$, the usual translation of ERD to relational database is assumed (e.g., [1]). We further assume all relation names are prefixed with the corresponding database name – DB$_1$::`staff` for instance.

An important issue in database interoperability is in resolving semantic heterogeneity, or what is referred to more generally as the matching process [9].

Past approaches have classified various types of heterogeneity intuitively. Here, we can characterize them more formally. First, we call an XTQL-query $Q$ *standard* if it has the form:

$$(T_1) \cup ... \cup (T_m), m \geq 1$$

where each $T_i$ is a T-query. $m$ is called the *degree of* $Q$. Suppose $D$ is an object-database. We denote by $\mathcal{M}(D)$ the set of all XRDB (MDB,ODB) such that ODB = $D$.

**Definition (Degree of Dissimilarity).** Suppose $D_1, ..., D_n$ are object-databases, and $R_1, ..., R_n$ are union-compatible output relations from a set of $n$ ordinary relational algebra queries over each of the respective databases. Without loss of generality, we assume that the relation names among the $D_i$'s are standardized apart. Let $\mathbf{R} = \cup_i^n R_i$, and let $\mathbf{Q}$ denote the set of all standard meta-queries over any XRDB $M \in \mathcal{M}(D_1 \cup ... \cup D_n)$ that evaluate to $\mathbf{R}$.

We say that the *degree of dissimilarity* among $D_1, ..., D_n$ with respect to $\mathbf{R}$ is $m/n$, where $m$ is the minimal-degree of all queries in $\mathbf{Q}$.

The idea of the definition is as follows. $D_1$ through $D_n$ represent data sources that contain semantically related data, but they may be designed differently. What dissimilarity attempts to capture is the question: if $R_1, ..., R_n$ are results that are deemed conceptually equivalent from the $n$ data sources, then how hard is it to design a set of meta-relations over $D_1, ..., D_n$ so that the minimal number of T-queries can be used to return precisely the union of the $R_i$'s? Note that minimal dissimilarity is $1/n$, since at least one T-query is required. We apply the definition.

An instance of semantic heterogeneity occurs with name conflicts [8]. Conventional methods for addressing name conflicts involves the creation of mappings between names. Here, rather than assuming a direct mapping between names in the two databases, we consider a mapping of design-specific names to a set of agreed upon keywords (or context-data [13]). An example would be that both the `no` attribute of `staff` and the `number` attribute of `engineer` map to the keyword `id`. Another example is for `employee` to be a keyword for both `staff` and `engineer`. Each relation and attribute name may map to multiple keywords. We introduce two meta-relations to manage keyword mappings: `mAttribute_synonym` and `mRelation_synonym`. A tabular representation of the meta-level information just described is as follows.

mAttribute_synonym

| mRel | mAtt | mKeyword |
|---|---|---|
| DB1:staff | no | id |
| DB2:engineer | number | id |
| DB1:staff | sname | name |
| DB2:engineer | name | name |

mRelation_synonym

| mRel | mKeyword |
|---|---|
| DB1:staff | employee |
| DB2:engineer | employee |

The data in `mAttribute_synonym` also show that DB$_1$:`staff.sname` and DB$_2$:`engineer.name` are semantically equivalent to `name`.

Once the metadata have been established, simple name conflicts (i.e., naming two semantically identical relations differently) can be addressed quite easily using the following parameterized query (declared as a cursor):

```
EXEC SQL declare resolve_name_conflict
CURSOR FOR
  UNION (SELECT $a
           FROM $r)
  WITH [relS.mRel r, attS.mAttr a]
  BASED ON
   (SELECT attS.mAttr, relS.mRel
    FROM  relation_synonym relS,
          mAttribute_synonym attS
    WHERE relS.mKeyword = :myRelation
      AND attS.mRel = relS.mRel
      AND attS.mKeyword = :myAttribute)
```

A mediated query to find all employee names can be obtained from the above query by assigning the values `employee` and `name` to the two host variables.

```
/* set the parameters */
myRelation = "employee";
myAttribute = "name";
/* next execute the statement */
EXEC SQL open resolve_name_conflict;
```

The same query can be used to resolve all similar name conflicts — e.g., `staff.no` versus `engineer.number` — by simply assigning the appropriate values to the host variables prior to executing the statement.

In XTQL, the same query would be written as the following parameterized expression. In the query, $rs$ stands for the meta-relation `mRelation_synonym`, and $as$ is an abbreviation for the meta-relation `mAttribute_synonym`.

$$Q(R, A) = \cup\pi_{\$\texttt{mAtt}}(\$\texttt{mRel})( \pi_{\texttt{mRel},\texttt{mAtt}}( \sigma_{rs.\texttt{mKeyword}=R \wedge rs.\texttt{mRel}=as.\texttt{mRel} \wedge as.\texttt{mKeyword}=A}(rs \times as)))$$

The degree of dissimilarity of $DB_1$ and $DB_2$ in this case is $1/2$. In general, to retrieve data from sources that differ only in names, the dissimilarity is $1/n$ where $n$ is the number of sources. Indeed, the dissimilarity is the same if the sources are identical in structure and name; a standard XTQL-query of degree 1 is required regardless of how the meta-relations are designed.

A more complex interoperability issue concerns structural conflicts. An instance of such conflict in the above example occurs with customer. Specifically, the example exhibits a type conflict, in which DB$_1$ represents customer as an attribute, while DB$_2$ represents customer as a relation (after the translation from ERD).

A query to find all project and customer combinations among the two databases requires knowing first of all, that DB$_1$:`project.cname` represent the customer name. This can be addressed via the keyword approach discussed above.

We assume that the meta-relations `mAttribute_synonym` and `mRelation_synonym` have been updated with the following data.

mAttribute_synonym

| mRel | mAtt | mKeyword |
|---|---|---|
| DB1:project | pname | name |
| DB1:project | cname | customer name |
| DB2:project | name | name |
| DB2:client | name | name |

mRelation_synonym

| mRel | mKeyword |
|---|---|
| DB2:client | customer |
| DB2:project | project |
| DB1:project | project |

Moreover, assume that the `mForeign_key` relation contains the following tuple:

```
< mSourceRel:DB2:project,
  mTargetRel:DB1:client,
  mSourceAtt:cid, mTargetAtt:cid >
```

The query shown in Figure 4 resolves the above attribute-relation conflicts. Note that the query resolves all attribute-relation conflicts under two additional assumptions: foreign keys in the database are composed of single attributes, and the design that represents the concept of interest as two relations are in a many-to-one relationship. The latter assumption allows the two relations to be joined.

The degree of dissimilarity here is $2/2$. This reflects that for each data source, a T-query that reflected the particular structure of the source had to be formulated. Observe that regardless of how the meta-relations are designed, it is not possible to pose a standard XTQL-query of degree 1 that returns the same results *unless* `project.name` and `customer.name` reside in the same relation in $DB_2$.

```
EXEC SQL DECLARE attr_rel_conflict CURSOR FOR
   [ UNION (SELECT rel1.$A, rel2.$B
            FROM  $R1 rel1, $R2 rel2
            WHERE rel1.$K1 = rel2.$K2)
     WITH [as1.mAtt A, as2.mAtt B,
           fk.mSourceRel R1, fk.mSourceAtt K1,
           fk.mTargetRel R2, fk.mTargetAtt K2]
     BASED ON
      (SELECT as1.mAtt, as2.mAtt,
               fk.mSourceRel, fk.mSourceAtt,
               fk.mTargetRel, fk.mTargetAtt
       FROM mForeign_key fk,
            mRelation_synonym rs1,
            mRelation_synonym rs2,
            mAttribute_synonym as1,
            mAttribute_synonym as2
       WHERE rs1.mKeyword = :myEntity1 AND
             rs2.mKeyword = :myEntity2 AND
             fk.mSourceRel = rs1.rel AND
             fk.mTargetRel = rs2.rel AND
             rs1.mRel = as1.mRel AND
             as1.mKeyword = :myAttribute1 AND
             rs2.mRel = as2.mRel AND
             as2.mKeyword = :myAttribute2)
    ]
    UNION
    [ UNION (SELECT rel.$A, rel.$B
             FROM $R AS rel)
      WITH [rs.mRel R, as1.mAttr A, as2.mAttr B]
      BASED ON
       (SELECT rs.mRel, as1.mAttr, as2.mAttr
        FROM  mRelation_synonym rs,
              mAttribute_synonym as1,
              mAttribute_synonym as2
        WHERE rs.mKeyword = :myEntity1 AND
              as1.mRel = rs.mRel AND
              as1.mKeyword = :myAttribute1 AND
              as2.mRel = rs.mRel AND
              as2.mKeyword = :myEntity || ' ' ||
                            :myAttribute2)
    ]
```

**Figure 4: Query to resolve attribute-relation conflicts**

Some other types of conflicts and their degrees of dissimilarity can be classified. For instance, cardinality constraint conflicts occurs when in one data source $D_1$, two relations $E_1$ and $E_2$ are in a many-to-many relationship $R_1$ while in a second data source $D_2$, the same two relations are in a many-to-one relationship $R_2$. Conventional design wisdom specifies that a $R_1$ should be represented as a relation of $D_1$ while $R_2$ is absorbed as an attribute in $E_1$ of $D_2$. Clearly, to find all combinations of $E_1$ and $E_2$ in both data sources also require a standard XTQL-query of degree 2.

These examples illustrate how different types of heterogeneity can be addressed via a relatively small set of XTQL queries over a set of appropriately designed meta-relations, thus avoiding the need for explicitly constructing mediated schemas. In each query, the metadata retrieved from the meta-query is used to dynamically construct the appropriate source-specific query. The approach is applicable in both a peer-to-peer data management setting [2], or a more traditional global mediated systems setting. The former has the advantage that the meta-relations can be more narrowly designed – they need not be one size fits all.

## 6. RELATED WORK

*SchemaSQL.* With respect to motivation, the closest related work to XRDB is the fundamental work on SchemaSQL by Lakshmanan, Sadri, and Subramanian [5, 6]. SchemaSQL aims to facilitate interoperability among relational databases that are semantically related but structurally dissimilar. The language extends SQL with range specification expressions that permit database, relation, and attribute names to be queried in order to reconcile structural variations among data sources.

Motivation aside, the details of the query languages in the two approaches are very different. In addition, SchemaSQL allows restructing of data and metadata while XRDB does not. This is conscious choice in the design of XRDB and it relates to a basic difference in philosophy: In SchemaSQL, metadata and data are treated uniformly, while in XRDB, metadata are clearly delineated from object-data. The XRDB philosophy is derived largely on the principles of self-describing data (e.g., XML), in which the status of metadata and data are distinct. A consequence of this separation is that the idea of restructuring does not arise in XRDB; neither data in object-relations can be promoted to metadata, nor data in meta-relations (i.e., metadata) can be demoted to object-data.

*The Ross Algebra.* As with SchemaSQL, the expansion operator introduced by Ross in [12] facilitates the restructuring of metadata and data. A similarity to XTQL is the dependence on arities of the computed relations. As a comparison, an equivalent query to Query 2 over the example relations of Figure 1 can be posed in the Ross Algebra (RA) as follows.

$$\pi_2(\alpha^2(\pi_{\texttt{mRel}}(\sigma_{\texttt{mKeyword = "SSN"}}(\texttt{mKeywords}))))\cup$$
$$\pi_2(\alpha^3(\pi_{\texttt{mRel}}(\sigma_{\texttt{mKeyword = "SSN"}}(\texttt{mKeywords}))))$$

Here, $\alpha^2$ and $\alpha^3$ are the expansion operators defined in [12]. The example also highlights another difference between XTQL and RA for performing data integration:[2] If we consider RA under the assumption that meta-relations and object-relations are separated, then an RA query to integrate object-data first reorganizes metadata and object-data into a single object-relation before appropriate conditions are applied. In contrast, a T-query in XTQL takes metadata as input, and its output is a set of appropriately structured SQL (or relational algebraic) queries that can then be applied directly to object-relations. Diagrammatically, the difference is illustrate in Figure 5. In practice, bypassing the restructuring step should lead to better performance.[3]

*Heterogeneity.* Section 5 shows how XRDB can be used to resolve heterogeneity among databases. There is an abundance of research that deals with this very broad topic, however, and even the concept of heterogeneity encompasses a variety of interpretations. While no attempt is made to address all issues related to data heterogeneity, the examples of Section 5 illustrate that XRDB can be effective in addressing what Seligman and Rosenthal refer to as heterogeneous attribute representations and semantics, and heterogeneous schemas [14].

## 7. DESIGN AND A PROTOTYPE IMPLEMENTATION

The objective of the prototype is to study the feasibility of a non-invasive XRDB design, including efficiency and performance issues. Figure 6 shows the organization of the system. The design makes use of a conventional relational DBMS (Oracle) to take advantage of available transaction processing facilities and the query

[2]It should be noted that data integration is not the primary motivation of Ross's work.

[3]This is a research issue under investigation.

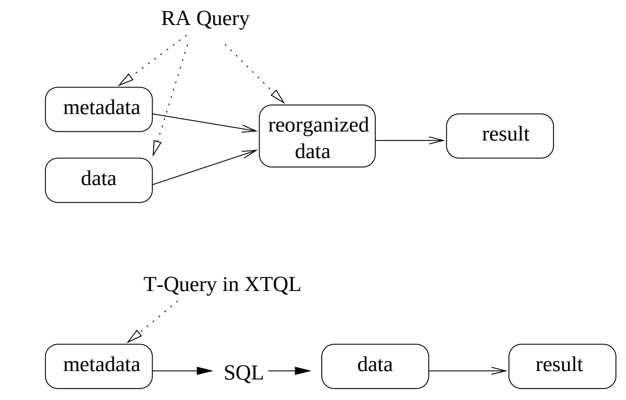

**Figure 5: RA compared to XTQL**

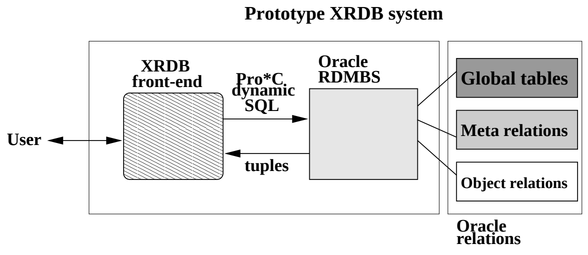

**Figure 6: The prototype XRDB system**

optimizer. The user interacts with the XRDB front-end through a simple command line interface. Each command is processed first via a syntax analyzer built using Lex/Yacc. For each correct statement, one or more ordinary SQL statements are then created and processed by the back-end processor.

An important design consideration is to introduce new types (meta-datatypes) that allow the XRDB system to differentiate the meanings of particular values in meta-relations from ordinary values. Below we discuss meta-datatypes associated with relation, attribute, key and foreign key constraints. These form the core meta-datatypes. Other meta-datatypes that are included in the current XRDB design but not discussed here include uniqueness and range constraints.

## 7.1 Meta-Datatypes

Meta-datatypes are grouped into two classes: `ID` and `Ref`. Conceptually, values of both classes are name of objects in the ODB. However, values belonging to a `Ref`meta-datatype needs to reference a corresponding `ID`meta-datatype. The uniqueness of `ID`meta-datatypes comes from its operational semantics. This will be discussed in Section 7.2.

*relationID.* `relationID` is a meta-datatype whose values correspond to names of object-relations. A value of type `relationID` is a unique identifier picking out a single relation in the object-database. A simple example schema of a meta-relation that uses this type, written in SQL DDL notation, is shown next.

```
CREATE TABLE mRelation (
   mName    relationID, /* a meta-type */
   PRIMARY KEY (mName)
);
```

Declaratively, a tuple `("D:employee")` in `mRelation` indicates that `employee` is the name of a relation in the database `D`.

**Restriction:** A base schema that has an attribute of type `relationID` can have no other meta-datatype attribute within the same relation.[4]

*relationRef.* Depending on applications, additional metadata about relations may be useful. An example from earlier discussion is, we may wish to maintain information about whether a given object-relation is derived from an entity or a relationship in a conceptual ER diagram. As another example, assuming a database design is a collaborative effort, we may wish to annotate object-relations with the designer of the relation. Rather than representing all these information about relations in a single meta-relation — which easily leads to a poor database design, separate meta-relations may be created. To facilitate the creation of such meta-relations, we introduce the meta-datatype `relationRef` whose domain consists of the same set of values as `relationID`. Any value of the type, however, must exist in a `relationID` attribute in another meta-relation. This requirement can be enforced via foreign keys. The following example schema illustrates the meta-relation for capturing ERD design information.

```
CREATE TABLE mERD_information (
   mRel      relationRef,
   mERD_type string,
   PRIMARY KEY (mRel),
   FOREIGN KEY (mRel)
     REFERENCES mRelation(mName)
);
```

Assume that the object-relation `employee` exists. We can express the meta-information that it is derived from an entity of an ERD via the tuple `("employee", "entity")`.

*attributeID and attributeTypeRef.* Similar to `relationID`, `attributeID` is a meta-datatype whose values correspond to attributes of object-relations. In this sense, values of `attributeID` are understood only in the context of a `relationRef` attribute; any meta-relation with an attribute of type `attributeID` must also possess an attribute of type `relationRef` which gives the target relation for the field. The most straightforward use of this meta-datatype is in the meta-relation `mAttribute` described in Section 2:

```
CREATE TABLE mAttribute (
   mRel    relationRef,
   mAtt    attributeID,
   mType   attributeTypeRef,
   PRIMARY KEY (mRel,mAtt),
   FOREIGN KEY (mRel)
     REFERENCES mRelation(mName)
);
```

The meta-datatype for the attribute `mType` is `attributeTypeRef`, and can be used to reference types in SQL:1999 as well as user-created types discussed below.

**Restriction:** A base schema that has an `attributeID` attribute must have an attribute of type `relationRef`, an attribute of type `attributeTypeRef`, and no other meta-datatype attribute.

*attributeTypeID.* In addition to built-in datatypes, user-defined types may be specified with meta-datatype `attributeTypeID`. This is useful for creating more descriptive type names (e.g., `nameType` instead of `string`), or for restricting a given base type (e.g., enumeration type). The example below illustrate a simple use.

[4]A base schema is one that has been defined explicitly through the SQL `CREATE TABLE` statement.

```
  CREATE TABLE mUser_type (
    mBase    attributeTypeRef,
    mName    attributeTypeID,
    PRIMARY KEY (mName)
  )
```

**Restriction:** A base schema that has an `attributeTypeID` attribute must have an attribute of type `attributeTypeRef` and no other meta-datatype attribute.

*attributeRef.* The need for different types of references to attributes in object-relations arises, and `attributeRef` represents the most basic meta-datatype for referencing attributes of relations. An example of another type of reference is to identify associations with a foreign key constraint. A common and important characteristic among all different types of attribute references is that, as with `attributeID`, a field reference must occur in the context of a particular "owning" relation. Thus, whenever an attribute of type `attributeRef` occurs in a meta-relation, a corresponding `relationRef` is required.

An example of a meta-relation that might include an attribute of this type is to restrict the syntactic forms of various string type attributes (e.g., SSN, Phone, etc.), discussed in Section 2:

```
CREATE TABLE mAttribute_form (
   mRel       relationRef,
   mAtt       attributeRef,
   mPattern   string,
   PRIMARY KEY (mRel,mAtt),
   FOREIGN KEY (mRel,mAtt) REFERENCES
      mAttribute(mRel,mAtt)
);
```

**Restriction:** A base schema that has an `attributeRef` attribute must have exactly one attribute of type `relationRef`.

*keyConstraintID.* The meta-datatype `keyConstraintID` is included in XRDB to capture primary key constraints. Its use in a relation requires the existence of an attribute reference (and its associated relation reference), since a key constraint exists only in the context of an attribute, or a group of attributes. A simple meta-relation that can be used to specify the primary key of various object-relations follows.

```
CREATE TABLE mPrimary_key (
   mRel        relationRef,
   mAtt        attributeRef,
   mPk_name    keyConstraintID,
   PRIMARY KEY (mRel,mAtt),
   FOREIGN KEY (mRel,mAtt) REFERENCES
      mAttribute(mRel,mAtt),
   CONSTRAINT pk_unique
      UNIQUE (mAtt,mPk_name)
);
```

Suppose we have an `employee` object-relation. Then the following `mPrimary_key`-tuple specifies that its primary key is the attribute `SSN`.

```
("employee","ssn","employee_pk")
```

Observe that the key of the meta-relation `mPrimary_key` consists of the attributes `mRel` and `mAtt`. This is necessary in case of composite keys. E.g., the combination of first and last name is used as the primary key for `employee` can be indicated via the two tuples:

```
("employee","firstname","employee_pk")
("employee","lastname","employee_pk")
```

The interpretation of the two tuples is that the constraint `employee_pk` is composed of both attribute references.

Alternatively, the primary key may consists of the attributes `mAtt` and `mPk_name` (hence the reason for the constraint `pk_unique`). The important point here is that there is a one-to-one relationship between `mRel` and `mPk_name`. That is, the functional dependencies `mRel` $\rightarrow$ `mPk_name` and `mRel` $\leftarrow$ `mPk_name` both hold. This ensures first of all that one and only one primary key exists for a given relation, and secondly that the same key name cannot be used for more than a single object-relation.

**Restriction:** A base schema that has a `keyConstraintID` attribute must have an attribute of type `relationRef`, an attribute of type `attributeRef`, and no other meta-datatype attribute.

*foreignKeyConstraintID.* Establishing foreign key constraints require two attributes: a referencing attribute, a referenced attribute, and their associated relations. We introduce the following meta-datatypes to complement `foreignKeyConstraintID`.

***sourceAttributeRef*** the referencing attribute

***targetAttributeRef*** the referenced attribute

***sourceRelationRef*** the relation associated with the referencing attribute

***targetRelationRef*** the relation associated with the referenced attribute

The first two are subsets of `attributeRef`, while the last two are subsets of `relationRef`. The simplest meta-relation for representing foreign key constraint is `mForeign_key` described previously. Its schema can be defined as follows.

```
CREATE TABLE mForeign_key (
   mSourceRel   sourceRelationRef,
   mSourceAtt   sourceAttributeRef,
   mTargetRel   targetRelationRef,
   mTargetAtt   targetAttributeRef,
   mFk_name     foreignKeyConstraintID,
   PRIMARY KEY
      (mSourceAtt,mTargetAtt,mFk_name),
   FOREIGN KEY (mSourceRel,mSourceAtt)
      REFERENCES mAttribute(mRel,mAtt),
   FOREIGN KEY (mTargetRel,mTargetAtt)
      REFERENCES mAttribute(mRel,mAtt),
);
```

Similar to composite keys, foreign keys consisting of multiple attributes require multiple tuples in the meta-relation to capture the constraint.

**Restriction:** A base schema with a `foreignKeyConstraintID` attribute must have attributes of type `sourceAttributeRef`, `targetAttributeRef`, `sourceRelationRef`, and `targetRelationRef`, and no other meta-datatype attribute. Conversely, each of the four attributes that references relations and attributes to relation must occur in a meta-relation that has an attribute of type `foreignKeyConstraintID`.

## 7.2 Mode

A question that needs to be addressed in the design of an XRDB system is, what is the operational semantics of insert, deletes, and updates that involve values belonging to ID meta-datatypes? Recall from the discussion of the previous section, a value occurring in an ID meta-datatype attribute indicates the existence of some object in the ODB, the object database component of an XRDB. While operationally, it would be natural to create/drop/modify the associated ODB object upon insert/delete/update, an equally important use of XRDB is as a system for the development of multi-database systems based on existing relational databases. In the latter case, an existing database becomes part of the ODB, with the MDB built "after the fact".

To accommodate both uses of XRDB, we allow a *mode* to be declared when an XRDB is initially created. The possible values for the mode are `active` and `passive` (the default value). If declared as `active`, then an insert into an ID meta-datatype attribute not only adds values to the meta-relation, but it has the additional side effect of creating the corresponding database object in the ODB. On the other hand, a `passive` declaration indicates that the XRDB system is responsible only for "connecting" any inserted value of an ID meta-datatype attribute with an existing ODB object. Thus, the `active` mode is appropriate when creating an XRDB from scratch, with tightly coupled MDB and ODB.

## 7.3 Query Processing

Since the XRDB query language is essentially SQL, queries that do not contain metadata facilities (i.e., T-expressions) are passed through to the native relational database system.

For each meta-datatype $T$ (e.g., relation name, attribute name), XRDB maintains a systems table Obj$T$ (a meta-meta-relation) that contains all values of $T$ that exist in the MDB. Since Oracle and other ordinary relational database systems do not recognize meta-datatypes, attributes of these types are stored as strings, along with any additional information that give these values their special status. A meta-attribute systems table is maintained to manage information on each meta-attribute; an example to illustrate its structure is shown in Figure 7.

Meta-Meta-Relation

| relation | attribute | type |
|---|---|---|
| mKeywords | mRel | relationRef |
| mKeywords | mAtt | attributeRef |
| mKeywords | mKeyword | string |

**Figure 7: An example XRDB system table**

When given a T-query

```
[UNION|INTERSECT] ( <text1> )
WITH [ <meta-attribute-list> ]
BASED ON ( <text2> )
```

contents of `text1` and `text2` are placed into two string variables, `MetaQ` and `DriverQ`, respectively. The XRDB front-end first executes `DriverQ` as a dynamic SQL statement. Then, the front-end iterates through the results and instantiates the list of variables in `meta-relation-list` to replace the corresponding parameters in `MetaQ`. Each instantiation of `MetaQ` is then passed into Oracle, with results of all instances collected (unioned or intersected) before returning to the user. The above evaluation algorithm is currently implemented and all of the example queries from previous sections have been tested.

## 8. CONCLUSION AND FUTURE RESEARCH

The work described here addresses a long-standing distributed data management problem – to find a scalable solution for database integration and interoperability. The scalability of the XRDB approach comes from the explicit and separate representation of metadata, as meta-relations. This allows the separation of meta-level information from programs that rely on that information to provide data translation among heterogeneous sources.

Clearly, the onus of creating a multi-database system out of existing data sources now falls on the development of appropriate meta-relations. Much like the efforts of various domain-specific standardization projects in XML (e.g., MathML), task-specific standardization of meta-relations will be required in order for XRDB to achieve its intended objectives. While the standardization efforts can be significantly simplified in a peer-to-peer data management setting [2] (since meta-relations can be more narrowly designed), interesting research issues on meta-relation designs arise nevertheless. In particular, the challenges in designing to solve semantic heterogeneity are very different from those to solve schematic heterogeneity — viz., level 3 versus level 4 heterogeneity [14]. A more systematic study of these issues is ongoing.

Compared to previously proposed approaches, an intuitively appealing quality of XRDB is the following. Metadata management is handled separately and entirely within the relational database setting. While the discussion in this paper assumes that existing data sources are relational databases, an intriguing question is, can relational metadata be used to integrate non-relational data sources. That is, given a data source $D$ in any data model, can relational metadata be created for $D$ in such a way that all accesses to $D$ are achieved through relational query languages?

The definition of schema dissimilarity is currently a naive one. More sophisticated measure of dissimilarity may be useful to accurately capture the intuition of dissimilarity among data sources. For instance, one possibility is to consider, in addition, the size of the meta-relations computed by the meta-query of each T-query.

## 9. REFERENCES

[1] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley, third edition, 2000.

[2] A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatorinov. Schema mediation in peer data management systems. In *19th International Conference on Data Engineering*, March 2003.

[3] P. C. Kanellakis. Elements of relational database theory. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science: Volume B: Formal Models and Semantics*, pages 1073–1156. Elsevier, Amsterdam, 1990.

[4] A. Kementsietsidis, M. Arenas, and R. J. Miller. Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In *Proceedings of SIGMOD*, 2003.

[5] L. V. S. Lakshmanan, F. Sadri, and I. N. Subramanian. SchemaSQL: A language for interoperability in relational multidatabase systems. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, and N. L. Sarda, editors, *22nd International Conference on Very Large Databases (VLDB 1996)*, pages 239–250, Bombay, India, 1996.

[6] L. V. S. Lakshmanan, F. Sadri, and S. N. Subramanian. Schemasql: An extension to sql for multidatabase interoperability. *ACM Transactions on Database Systems (TODS)*, 26(4):476–519, 2001.

[7] M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, 2002. Invited tutorial.

[8] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. second edition, 1999.

[9] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal: Very Large*

*Data Bases*, 10(4):334–350, 2001.

[10] A. Rosenthal and L. Seligman. Scalability issues in data integration. Prsented at the AFCEA Federal Database Colloquium, 2001.

[11] A. Rosenthal and L. Seligman. Decentralized development without a global blueprint. MITRE Technical Report, 2002.

[12] K. A. Ross. Relations with relation names as arguments: Algebra and calculus. In *Proc. of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 346–353, San Diego, CA, 1992.

[13] E. Sciore, M. Siegel, and A. Rosenthal. Using semantic values to facilitate interoperability among heterogeneous information systems. *ACM Transactions on Database Systems (TODS)*, 19(2):254–290, 1994.

[14] L. Seligman and A. Rosenthal. The impact of xml on databases and data sharing. *IEEE Computer*, 34(6):59–67, 2001.

[15] V. Sugumaran and V. C. Storey. An ontology-based framework for generating and improving database design. In *Natural Language Processing and Information Systems: 6th International Conference on Applications of Natural Language to Information Systems*, pages 1–12. Springer, 2002.

[16] V.S.Subrahmanian *et. al.* Hermes: A heterogeneous reasoning and mediator system. http://www.cs.umd.edu/projects/hermes/overview/papers, 1998.