# A Data Preparation Framework based on a Multidatabase Language

Kai-Uwe Sattler Eike Schallehn
Department of Computer Science
University of Magdeburg
P.O.Box 4120, 39016 Magdeburg, Germany
{kus|eike}@iti.cs.uni-magdeburg.de

## Abstract

*Integration and analysis of data from different sources have to deal with several problems resulting from potential heterogeneities. The activities addressing these problems are called data preparation and are supported by various available tools. However, these tools process mostly in a batch-like manner not supporting the iterative and explorative nature of the integration and analysis process. In this work we present a framework for important data preparation tasks based on a multidatabase language. This language offers features for solving common integration and cleaning problems as part of query processing. Combining data preparation mechanisms and multidatabase query facilities permits applying and evaluating different integration and cleaning strategies without explicit loading and materialization of data. The paper introduces the language concepts and discusses their application for individual tasks of data preparation.*

## 1. Introduction

Preparing data is an important preprocessing step for data analysis not only as part of data warehousing but also for data mining. The main reason is that the quality of the input data strongly influences the quality of the analysis results. In general, data preparation comprises several subtasks: selection, integration, transformation, cleaning as well as reduction and/or discretization of data. Often these tasks are very expensive. So, sometimes it is stated that 50-70 percent of time and effort in data analysis projects is required for data preparation and only the remaining part for the actual analysis phase.

Currently, there are several tools available for different preparation tasks, partially also as integrated suites, e.g. products like Microsoft Data Transformation Service or Oracle DataWarehouse Builder. However, these tools are based mainly on a batch principle: the input data are read from the database, processed in main memory according to the tool-specific task and finally written back to the database in place or into a new table. As a consequence either the initial dataset is overwritten or for each preparation step an additional copy is required. If the data come from different external sources an auxiliary loading step has to precede the preparation. Moreover, the preparation tasks are specified operationally, i.e. have to be implemented by user-defined procedures and scripts. Together, this raises problems especially, if the data that is subject of analysis is coming from distributed and possibly heterogeneous sources, large datasets have to be examined or it is not known at the beginning of the analysis which preparation and/or analysis techniques are really required.

Therefore, a better approach should enable to apply major preparation steps in a more declarative manner, i.e. as part of queries on virtual integrated datasets. In this way, the preparation tasks could profit from the capabilities of a database management system, e.g. optimization of queries, parallelization and memory management. In addition, if this approach would be combined with a multidatabase or gateway technology, an interactive and explorative preparation and analysis without an explicit materialization is possible. Based on this observation we investigate in this paper to which extent important and widespread problems of data preparation are solvable by applying standard SQL features as well as language features proposed recently in the research literature and implemented in our multidatabase query language FRAQL. This language provides powerful query operations addressing problems of integration and transformation of heterogeneous data [25] and therefore, it is a suitable platform for building up a framework for data preparation.

The paper is structured as follows. In Section 2 we discuss the process of data preparation, assign subtasks and common techniques to the individual steps. Section 3 introduces query language constructs supporting integration and transformation of data. Cleaning techniques are presented in Section 4 and problems of reduction and discretization are addressed in Section 5. Section 6 discusses related work. Finally, Section 7 concludes the paper, gives an overview to the usage of FRAQL as part of a workbench for data integration and analysis and points out to future work.

## 2. Major Tasks in Data Preparation

In Section 1 we already mentioned the importance of data preparation for analysis and mining. In the following we will discuss the individual tasks of this process. For a better understanding we first sketch the overall process of analysis. There are various views on the analysis process depending on the application domain, the required data and analysis methods. One view is proposed by the CRISP-DM consortium – the Cross-Industry Standard Process for Data Mining – a reference model describing the individual phases of the data mining process [8]. Following CRISP-DM a data mining project consists of six phases. The first phase *business understanding* addresses the understanding of the project objectives and requirements from a business perspective, the second phase *data understanding* focuses on collecting, describing and exploring data for further examination. The third step is *data preparation*, followed by the *modeling* phase where modeling techniques (e.g., classification, clustering etc.) are selected and applied as well as parameter sets are adjusted. Next, the models have to be evaluated in order to fulfill the quality requirements (*evaluation*). Finally, the *deployment* phase consists of tasks like report generation and documentation.

As part of this process data preparation comprises all activities required for constructing the input dataset for the modeling from the initial (raw) dataset [14]. This includes

**data selection:** Identify data relevant and useful for analysis and select the required attributes and tuples.

**data integration:** Combine data from multiple sources and/or relations and construct new tuples and values.

**data transformation:** Transform the data on structural as well as syntactic level meeting the requirements of the analysis tools, for example normalization and scaling, derive or compute new values and/or tuples.

**data cleaning:** Improve the data quality by selecting clean subsets of data, filling in missing values, removing noise, resolving inconsistencies and eliminating outliers.

**data reduction:** Obtain a reduced or compressed representation of the dataset in order to reduce the analysis effort by aggregating or discreticing data, reducing the dimensionality or the volume of data, e.g. deriving a sample.

In practice all these tasks – not only for preparation but also as part of the overall process – do not form a strict sequence. Rather the various techniques for data selection, preparation and analysis are applied in an iterative, interactive and explorative manner. Additionally, often several techniques are evaluated at the same time in order to find the most suitable method or to combine results from different methods.

Beside modeling, where efficient data access and query facilities are required, data preparation is one of the phases which could benefit mostly from extended query language support. Though standard SQL provides basic features for filtering and transforming data, several subtasks as cleaning, normalization, discretication and reduction require more advanced operations. In the following we discuss query primitives provided by FRAQL supporting these operations.

## 3. Integration and Transformation

Analyzing data from heterogeneous sources requires transparent access to these sources in order to retrieve, combine and transform the relevant data. This can be done either by loading the data physically in a special database (e.g. the staging area in a data warehouse environment or a dedicated analysis database) or by defining virtual views using a multidatabase system. This virtual integration approach is supported by our multidatabase language FRAQL. With respect to data integration and transformation a multidatabase language in general and in particular our language FRAQL provides the following benefit:

- transparent access to external sources,
- integrating relations from different sources via join and/or union operations,
- resolving description, structural and semantic conflicts with the help of renaming operations, conversion and mapping functions,
- advanced schema transformation, e.g. transposition of relations, as well as resolving meta-level conflicts,
- resolving data discrepancies, i.e. instance-level conflicts, using reconciliation functions and user-defined aggregates.

In the following we describe the corresponding FRAQL features in detail. For illustration purposes we will use two different scenarios. The first scenario describes a simplified data warehouse application and comprises a relation `SalesInfo` containing regional sales information on various products as well as a relation `AdvInfo` containing advertising costs for products and regions:

```
SalesInfo (Product, Prod_Group, Euro_Sales,
           Year)
AdvInfo (Product, Cost, Region, Year)
```

In the second scenario measurements are collected in a relation `Measurements` consisting of the actual values `X`, `Y` the geographical position of the point of measurement, a timestamp and a classifying attribute:

```
Measurements (X, Y, Longitude, Latitude,
              Time, Class)
```

## 3.1 Accessing and Integrating Heterogeneous Sources

FRAQL is an extension of SQL and provides access to relations hosted by external sources, e.g. full-featured database systems, relational structured documents as well as websites. The access is implemented via adapters which are loadable at runtime and translate queries as well as query results between the global multidatabase level and the source level.

FRAQL is based on an object-relational data model comprising user-defined types, views for integrating source relations and tables for managing global data. We distinguish two kinds of views: *import views* mapping relations from a source to global relations and *integration views* combining multiple views and/or tables. An import view is defined in terms of a user-defined type. Here, for each attribute the mapping to the source attribute can be defined either

- as simple renaming of the attribute,
- as value conversion by applying a user-defined function,
- or by using an explicit mapping table for converting values.

An *integration view* is a view on other global relations combined by using operators like union, $\theta$-join and outer join. These features provide a first solution for data transformation.

## 3.2 Schema Transformation

Based on global views further transformations on schema as well as on instance level can be applied. Schema level transformation in FRAQL includes – besides standard SQL operations like projection and join – restructuring via transposition.

Transposing means converting rows to columns and vice-versa. As an example, where this kind of restructuring is required, please consider the relations in Fig. 1.

| Product | Year | RegionA | RegionB | RegionC |
|---|---|---|---|---|
| $P_1$ | 2000 | 15 | 18 | 22 |
| $P_2$ | 2000 | 23 | 25 | 28 |

(a) Advertising

| Product | Cost | Region | Year |
|---|---|---|---|
| $P_1$ | 15 | RegionA | 2000 |
| $P_1$ | 18 | RegionB | 2000 |
| $P_1$ | 22 | RegionC | 2000 |
| $P_2$ | 23 | RegionA | 2000 |
| $P_2$ | 25 | RegionB | 2000 |
| $P_2$ | 28 | RegionC | 2000 |

(b) AdvInfo

**Figure 1. Example relations for transposition**

The relation `Advertising` contains the advertising costs of products for the given regions `RegionA` ... `RegionC`. For further processing, e.g. grouping and aggregation, a relation according to the definition of `AdvInfo` is required. For this purpose FRAQL provides the TRANSPOSE TO ROWS operation (as proposed in [6] and in a more general form in [17]): for each tuple of the input relation exactly $n$ tuples of the given structure are generated for the output relation. Therefore the above relation can be transformed as follows:

```
select *
from Advertising
transpose to rows
   (Product, RegionA, 'RegionA', Year),
   (Product, RegionB, 'RegionB', Year),
   (Product, RegionC, 'RegionC', Year),
   as (Product, Cost, Region, Year);
```

The inverse operation to TRANSPOSE TO ROWS is TRANSPOSE TO COLUMNS which takes a subset of the input relation containing the same value in a given column and constructs one output tuple with columns representing the different tuple values. In this way, the relation `AdvInfo` could be transformed back according to the structure of relation `Advertising`:

```
select *
from AdvInfo
transpose to columns
   (Cost as RegionA when Region='RegionA'
            RegionB when Region='RegionB'
            RegionC when Region='RegionC')
   on Product, Year
   as (Product, Year, RegionA,
       RegionB, RegionC);
```

The `on` part of the clause specifies the attributes used for identifying groups of tuples which are transposed to exactly one tuple. The operation is implemented similar to the GROUP BY operation, though the resulting groups are transformed into one tuple per group.

### 3.3 Data Transformation

Transformation on instance level means in fact converting values. Depending on the requirements of the following analysis tasks various approaches are feasible. First of all, simple value conversion can be performed using builtin or user-defined function (UDF). Particularly, the following functions are supported as builtin functions by FRAQL:

- string manipulation functions like Standard SQL functions `concat`, `substring` as well as `split` for splitting strings on a given delimiter,
- general-purpose conversion functions, e.g. `string2double`, `string2date`, etc.

A second common kind of transformation is normalization of (numeric) values. Often attribute values have to be normalized to lie in a fixed interval given by the minimum and maximum values. In this case the *min-max-normalization* is applied. Let $a$ be the attribute value, $min_o$ and $max_o$ the minimum and maximum value of $a$ and $min_n$ and $max_n$ the new interval boundaries for the normalized value $a'$:

$$a' = \frac{a - min_o}{max_o - min_o}(max_n - min_n) + min_n$$

This normalization can be implemented by the user-defined function `min_max_norm`:

```
double min_max_norm (double a,
            double omin, double omax,
            double nmin, double nmax) {
  return (a - omin)/(omax - omin)*
         (nmax - nmin) + nmin;
}
```

Another technique – the *z-score-normalization* – maps the values of an attribute into the interval $[-1 \ldots 1]$ using the mean and standard deviation of an attribute $a$:

$$a' = \frac{a - mean}{stddev}$$

This is implemented by a second UDF `zscore_norm` that requires the mean and standard deviation of the attribute values as parameters:

```
double zscore_norm (double a,
            double mean, double stddev) {
  return (a - mean)/ stddev;
}
```

Both normalization techniques require two scans on the input relation: a first one for computing the aggregates and a second one for the actual normalization step:

```
select avg(X), stddev(Y)
into :mean, :sdev from Measurements;
select zscore_norm(X,:mean,:sdev), Y, Class
from Measurements;
```

As already mentioned, all these operations can be applied to virtual relations, i.e. views, without affecting the original data. This simplifies the evaluation of different techniques for integration and transformation of heterogeneous data subject of analysis.

## 4. Cleaning

Integration and transformation are only the first steps in data preparation. Due to the often different origins of data resulting in different conventions for representation, errors like mistyping, missing values or simply inconsistencies, additional cleaning steps are necessary. In the following we will concentrate on four subproblems:

- identifying and reconciling duplicates,
- filling in missing values,
- detecting and removing outliers,
- handling noise in data.

### 4.1 Duplicate Elimination

Because of the tight connection to the integration problem a first solution for duplicate elimination based on the extended join and union operators in conjunction with reconciliation functions was already introduced in Section 3. However, the usage of reconciliation function is restricted to binary operations. Therefore, FRAQL provides an additional and more powerful solution based on extended grouping and user-defined aggregates. The GROUP BY operator supports grouping on arbitrary expressions: for each tuple of the input relation a value is computed from the grouping expression and based on this, the tuple is assigned to a group. Let us assume a UDF `region_code` computing the region from the geographical position. Using this function as a grouping criterion the regional averages of the measurements can be computed:

```
select avg(X), avg(Y), rc
from Measurements
group by
  region_code (Longitude, Latitude) as rc;
```

After the GROUP BY operation each group consists of tuples representing the same real-world entity. In the final step all the tuples of a group have to be merged in one item, for example by computing a value from conflicting attributes or by using the most up-to-date information. This can be implemented with the help of user-defined aggregates. A user-defined aggregate (UDA) function is implemented in FRAQL as a Java class with a predefined interface:

```
public interface UDA {
  void init ();
  boolean iterate (Object[] args);
  Object result ();
  Object produce ();
}
```

At the beginning, the method `init` is called. For each tuple the `iterate` method is invoked. The final result is obtained via the method `result`. Method `produce` is used for UDAs with early returns and will be introduced later. Because a UDA class is instantiated once for the whole relation the “state” of the aggregate can be stored. The following example shows a UDA returning the values of the newest tuple in a simplified form omitting some implementation details like value conversion.

```
class Newest implements UDA {
  Timestamp tstamp;
  double value;

  void init () { tstamp = 0; value = 0.0; }

  boolean iterate (double v,
                Timestamp ts) {
    if (ts > tstamp)
      { value = v; tstamp = ts; }
    return false;
  }

  double result () { return value; }
}
```

The Java class is registered in the FRAQL query system using the **`create aggregation`** command:

```
create aggregation newest (double,
   timestamp) returns double
   external name 'Newest';
```

Using both features together data discrepancies caused by semantically overlapping can be resolved as shown in the following example, where measurements from different stations are combined. The UDA collects all values of the attribute and returns the value associated with the most up-to-date measurement for a region:

```
select newest(X,Time), newest(Y,Time), rc
from Measurements1 union all
     Measurements2 ...  union all
     MeasurementsN
group by
  region_code (Longitude, Latitude) as rc;
```

Essentially both approaches are based on attribute equality. However, sometimes the same objects may be referred to in different sources by slightly different keys. In this case, duplicates can only be identified based on similarity criteria.

As shown before the process of duplicate elimination can be considered a two step process. In a first step we have to identify entities that possibly relate to the same real world object, which are reconciled in a second step. This fits quite well with the concept of grouping and aggregation known from relational databases, but while reconciliation can be accomplished by extensions to aggregation concepts including user-defined functions like shown before, similarity-based grouping raises various problems. By specifying a similarity criterion in an appropriate way we can establish a relation of pairwise similar tuples. This relation must be considered atransitive, so to establish the equivalence relation required for grouping we have to describe the construction of the groups.

This can be done by a generalized concept for user-defined aggregate functions. As an example, groups can be build based on pairwise similarity by

- considering the transitive closure of similar tuples,
- only considering maximal groups that contain tuples that are all pairwise similar, or
- building groups of tuples that are similar to a chosen representative of this group.

Contrary to equality-based grouping these approaches consider not only one tuple to derive a group membership, but instead have to analyse one tuple in the context of the whole input relation. As a consequence the interface for user-defined grouping functions (UDG) is as follows:

```
public interface UDG {
  void init (Object[] args);
  boolean iterate (int tuple_id,
               Object[] values);
  void finish ();
  int getGroupForTuple (int tuple_id);
}
```

Using the `init` method the function is initialized before the beginning of the grouping. Possible parameters, e.g. a description of the similarity criterion, can be specified when the function is registered. The function performs the grouping during iteration, if possible. All other actions required for building groups take place after iteration is finished. Not before that the FRAQL query engine can retrieve information about group memberships. For this purpose surrogate tuple and group identifiers are used internally.

A simple user-defined grouping function that groups products based on similarity of the product name from an integrated view of `SalesInfo` from various sources can be used as follows:

```
select product, sum(euro_sales)
from SalesInfo1 union all
     SalesInfo2 ...  union all
     SalesInfoN
group by similarity
     SameProductName(product);
```

The user-defined grouping function `SameProductName` could for example be initialized with the name of a comparison algorithm for strings and a threshold to compare the return value of this algorithm. For the following description of a simple grouping algorithm represented by the `SameProductName` function we assume that groups are build using the transitive closure of pairwise similar tuples. During iteration a tuple identifier and the product name is passed on to the function. The product name is compared to all product names from previous iteration steps. If the string comparison function returns a value above the threshold, the tuple is added to the group of the according previous tuple. If there are multiple matches, the related groups are merged. If there is no match, a new group containing this tuple is created. When the iteration is finished, FRAQL calls the `finish` method. In this case the only thing to be done here is to set a flag indicating that information about group memberships can be retrieved.

While this approach is quite flexible, it also has several disadvantages. The efforts for implementing user-defined grouping functions can become expensive when complex criteria or more advanced grouping algorithms are used. Furthermore, there is less opportunity to optimize the grouping when it is separated from query processing in the query evaluator. Therefore, we currently consider implementing some of the most useful concepts mentioned above as query language primitives using a combination of fuzzy logic, pre-defined and user-defined comparison functions for atomic attributes, and the proposed strategies for establishing an equivalence relation.

## 4.2 Missing Values

Considering the problem of missing values, e.g. NULL values, we can describe several possible solutions. First of all, the simplest approach is to remove the affected tuples using the SQL `not null` predicate:

```
select X, Y, Class, Time
from Measurements
where Y not null and Y not null;
```

However, this is problematic if the percentage of missing values is large. An alternative is to fill in the missing values with a value computed from others, e.g. the average or the median of the attribute or the attribute value of all tuples associated to the same class. The latter requires either grouping or clustering. In the following example, grouping is used to assign the average of the attribute value of a class to a missing value. The average values are stored in a temporary relation `tmp(Class, X)`:

```
insert into tmp (
  select Class, avg(X) as X
  from Measurements group by Class);

update Measurements
  set X = (select X from tmp
  where tmp.Class = Measurements.Class)
where X is null;
```

## 4.3 Handling Noise in Data

Noise in data is caused by random errors or variance. Among others there are two approaches for handling noise: binning or histogram methods and regression.

Using the *histogram method* the data are partitioned into bins or buckets based on a certain attribute. For this purpose the attribute domain is divided into $n$ intervals and each item value is associated to exactly one bucket.

We can distinguish between *equi-width* histograms, where each bucket has the same interval length and *equi-height* histograms, where each bucket contains the same number of items. The first one is simpler to construct and to use, the latter provides a better scaling.

In the following we will use a histogram relation

```
Histo (hmin, hmax, mean, cnt)
```

In this relation each tuple represents a bucket containing values $v$ with $hmin \leq v < hmax$, where $mean$ is the average and $cnt$ is the number of all values associated with this bucket.

Equi-width histograms can easily be constructed using the extended `group by`-operation. After choosing the number of buckets `nbuckets` and obtaining the domain boundaries of the attribute $x$:

```
select (max(X) - min(Y))/:nbuckets
into :width
from Measurements;
```

a histogram for the attribute $x$ containing the means of each bucket is constructed as follows:

```
insert into Histo (
  select id *:width, (id+1)*:width,
         avg(X), count(X)
  from Measurements
  group by floor (X / :width) as id);
```

After partitioning the data, they can be smoothed by different criteria, e.g. bucket mean, bucket median or bucket boundaries. In the following the bucket mean is used for smoothing the values of attribute $x$:

```
select mean, Y, Class, Time
from Measurements, Histo
where X >= hmin and X < hmax;
```

Creating equi-height histograms is slightly more complex. First, the frequency of all occuring values have to be determined.

```
select X, count(*)
from Measurements group by Y;
```

From the number of tuples and the number of buckets the height of each bucket can be computed. Next, we iterate over the sorted frequency table and collect items until the maximum bucket height is reached. The bucket boundary is computed from the value of the current items. Using only standard SQL the latter step cannot be formulated in a single query and has to be implemented by an external procedure.

A more elegant solution is possible using the advanced aggregation features provided by FRAQL. As introduced in [30] user-defined aggregation functions with *early returns* are a powerful mechanism for incremental evaluation of aggregates. FRAQL supports early returns by using a special method `produce` as part of the UDA interface. If the aggregate function wants to "return early", it returns the value `true` as result of the invocation of `iterate`. In this case the method `produce` is called from the query processor in order to obtain the intermediate aggregation value. As an example let us consider the implementation of a UDA for computing the moving average:

```
class MvAvg implements UDA {
  int num;
  double value;

  void init () { num = 0; value = 0.0; }
  boolean iterate (double v)
     {  num++; value += v; return true; }
  double produce ()
     { return value / num; }
  double result () { return produce (); }
}
```

UDAs with early returns simplify the construction of equi-height histograms. For this purpose three UDAs are required: `b_lb` for computing the lower boundary, `b_ub` for computing the upper boundary and `b_mean` for computing the bucket mean. These UDAs are now used in the following query for obtaining the histogram with buckets of height $height$ for the attribute $x$:

```
insert into Histo (
  select b_lb(X, cnt, :height),
         b_ub(X, cnt, :height),
         b_mean(X, cnt, :height), :height
  from ( select X, count(*) as cnt
         from Measurements
         group by X order by X asc ) );
```

The implementation of the UDAs is sketched for `b_ub` below:

```
class Bucket_ub implements UDA {
  int height;
  double upper, lower;
  ...

  boolean iterate (double val, int cnt,
                   int maxh) {
    height += cnt;
    if (height >= maxh) {
      lower = upper =
       compute_split (val, maxh);
      height = maxh - height;
      return true;
    } else {
      upper = val;
      return false;
    }
  }

  double compute_split (double val,
                        int maxh) {
    return maxh / height *
           (val - lower) + lower;
  }

  double produce () { return boundary; }
}
```

Please notice that this implementation is oriented to numeric attributes. For ordinal attributes an existing attribute value has to be chosen during the split instead of computing the new boundary.

Based on this histogram the data can be smoothed by the means of the buckets already shown for equi-width histogram.

## 4.4 Detecting Outliers

Histograms are also an appropriate tool for detecting outliers. Another frequent used technique for this purpose is regression. In *linear regression* also known as least square method the goal is to find a straight line modeling a two-dimensional dataset. This line $y = \alpha x + \beta$ is specified by the parameters $\alpha$ and $\beta$ which are calculated from the known values of the attributes $x$ and $y$. Let

$$\overline{x} = \frac{1}{n}\sum x_i \text{ and } \overline{y} = \frac{1}{n}\sum y_i$$

then holds

$$\alpha = \frac{\sum(x_i - \overline{x})(y_i - \overline{y})}{\sum(x_i - \overline{x})^2} \text{ and } \beta = \overline{y} - \alpha\overline{x}$$

Obtaining the parameters for this line is straightforward if a UDA `calc_alpha` is used:

```
select avg(X), avg(Y) into :xa, :ya
from Measurements;

select calc_alpha (X, Y, :xa, :ya)
into :alpha
from Measurements;

beta := ya - alpha * xa;
```

The parameters `alpha` and `beta` can now be used to remove data items far away from the regression line. For example, this can be simply decided based on the absolute distance or by removing $n$ percent of items with the largest distance. The first approach is expressible in a single query using an expression for computing the distance:

```
select * from Measurements
where abs(:alpha*X+:beta - Y) <
      :threshold;
```

The second criterion requires a mechanism for limiting the cardinality of a query result. In FRAQL this is supported by the LIMIT FIRST clause, which cuts the result set after the specified number of tuples. So, the query for removing 5 percent of tuples with the largest distance as outliers can be written as:

```
select X, Y,
       abs(:alpha*X+:beta - Y) as dist
from Measurements
order by dist
limit first 95 percent;
```

## 5. Data Reduction

The last subproblem of data preparation addressed in our framework is reduction. Sometimes the volume of data is too large for an efficient analysis. Therefore, we need a reduced representation of the dataset that produces nearly the same analysis results. Here, several solutions are possible, e.g. aggregating the data as usually performed in data warehouse environments or discarding non-relevant dimensions. The first approach is normally implemented using GROUP BY and aggregation, the second approach requires only projection. Two other approaches directly supported in FRAQL are sampling and discretization.

*Sampling* means to obtain a representative subset of data. Various kinds of sampling were developed recently. Currently, in FRAQL only random sampling as presented in [29, 22] is implemented. A sample of a relation or query result with the given size is derived with the help of the LIMIT SAMPLE clause:

```
select * from Measurements
limit sample 30 percent;
```

Obviously, the sampled data should be stored in a new relation for further efficient processing.

Sampling reduces the number of tuples by choosing a subset. In contrast, *discretization* is aimed to reducing the number of distinct values for a given attribute, particularly for analysis methods requiring discrete attribute values. Possible solutions for discretization are

- histogram-based discretization,
- discretization based on concept-hierarchies [13],
- entropy-based discretization [9].

Considering only the histogram-based approach numeric values could be replaced by a representative discrete value associated with the containing bucket as already discussed in Section 4.3. Here, both kinds of histograms are applicable. An alternative approach is to build histograms by hand taking domain knowledge into consideration, e.g. specify higher-level concepts like income class or price classification (cheap, medium-priced, expensive) together with the associated interval boundaries. Independent from the kind of construction, the histograms are used as illustrated already.

## 6. Related Work

In recent years there has been much effort on support for individual tasks of data preparation for data warehousing and data mining. Especially for the transformation problem there are many commercial tools available. These ETL (“extraction, transformation, loading”) tools [4, 1] address different kinds of data preparation, ranging from general purpose tools and tool suites to rather specialized tools for common formats. Moreover, so called auditing tools are able to detect and eliminate discrepancies. However, these tools support only limited set of transformations or the transformation has to be implemented by user-defined scripts or procedures. Therefore, they perform transformation in a batch-like manner not supporting an explorative and interactive approach.

Furthermore, several approaches addressing special data cleaning and transformation problems were proposed. AJAX [10] is a cleaning tools that supports clustering and merging duplicates. It is based on a declarative specification language. SERF [5] provides primitives for schema evolution and data transformation. An approach for schema transformation is described as part of SchemaSQL [17], a language offering restructuring operations. In [1] a declarative transformation language YATL is proposed and automation of the translation based on schema similarities is discussed. Methods to detect a maximum number of duplicate items (exactly or approximately) and to derive cleaning rules are proposed in [15]. An approach for resolving attribute value conflicts based on Dempster-Shafer theory, which assigns probabilities to attribute values, is described in [20]. An object-oriented data model where each global attribute consists of the original value, the resolved value and the conflict type is presented in [19]. These individual values are accessible by global queries. In addition, for each attribute a threshold predicate determining tolerable differences and a resolution function for an automatic conflict resolution can be defined. Another approach [26] introduces the notion of semantic values enabling the interoperability of heterogeneous sources by representing context information. An advanced application of statistical data analysis for deriving mapping functions for numerical data is described in [21].

Much recent work is dedicated to middleware supporting the integration of heterogeneous data, e.g. multidatabase languages like MSQL [12] and SQL/M [16], federated database systems like Pegasus [2] and IBM DataJoiner [28] as well as mediator-based sytems like TSIMMIS [11], Information Manifold [18], DISCO [27] and Garlic [3]. Pegasus offers a functional object-oriented data manipulation language called HOSQL with non-procedural features, DataJoiner is based on DB2 DBMS. In mediator systems such as TSIMMIS the mediator is specified by a set of rules. Each rule maps a set of source objects into a virtual mediator object. In this way, transformations are defined by appropriate rules. The problem of combining objects from different sources (object fusion) in mediators is discussed in [23].

A totally different approach is WHIRL [7], which deals with instance heterogeneity during integration. Here, textual similarity is used for computing joins between relations from different sources. This permits integration without normalization of values but is restricted to textual data.

There has been some work on query language extensions addressing preparation and analysis tasks. In [30] user-defined aggregates with early returns and their application for data mining is discussed. [6] presents SQL/MX, a query language offering features like transposing and sampling. Our approach is partially inspired by these query extensions but combines these with multidatabase language features.

## 7. Conclusions

Data preparation is an important task for data warehousing and data mining. As part of this process several problems have to be dealt with: data integration and transformation, detecting inconsistencies in data coming from multiple sources, removing outliers, suppressing noise and reducing data. However, writing dedicated routines addressing individual subproblems can be an expensive and error-prone process especially for information fusion [24] – the integration and analysis of data from heterogeneous sources.

In this paper we have presented a framework for data preparation based on the multidatabase query language FRAQL. The benefit of using a multidatabase language is the virtual integration combined with the ability to apply transformation and cleaning operations without copying and physically manipulating the initial dataset. So, it is possible to check various strategies for integration and cleaning with reduced effort. Thereby, the objective of our work is not to invent a new language or add general-purpose operations but rather to identify lightweight extensions dedicated to data preparation and analysis and add them to a SQL-like query language. Implementing these extensions as database primitives opens the possibility to profit from the inherent capabilities of database management systems.

The presented language extensions are implemented as part of the FRAQL query processor which forms the core of an engine for information fusion tasks. This engine offers an extensible set of operators for data preparation and analysis tasks. These operators are parameterizable and generate as well as execute FRAQL queries as presented in this paper. Combined with user interface and visualization tools a more interactive and iterative approach for data analysis is achieved. For future work the support of more advanced cleaning operations, e.g. similarity-based entity identification, as well as analysis and mining operations is planned.

## References

[1] S. Abiteboul, S. Cluet, T. Milo, P. Mogilevsky, J. Siméon, and S. Zohar. Tools for Data Translation and Integration. *IEEE Data Engineering Bulletin*, 22(1):3–8, 1999.

[2] R. Ahmed, P. D. Smedt, W. Du, W. Kent, M. Ketabchi, W. Litwin, A. Rafii, and M.-C. Shan. The Pegasus Heterogeneous Multidatabase System. *IEEE Computer*, 24(12):19–27, December 1991.

[3] M. Carey, L. Haas, P. Schwarz, M. Arya, W. Cody, R. Fagin, M. Flickner, A. Luniewski, W. Niblack, D. Petkovic, J. T. II, J. Williams, and E. Wimmers. Towards Heterogeneous Multimedia Information Systems: The Garlic Approach. In O. A. Bukhres, M. T. Özsu, and M.-C. Shan, editors, *Proc. RIDE-DOM '95, 5th Int. Workshop on Research Issues in Data Engineering - Distributed Object Management, Taipei, Taiwan*, pages 124–131. IEEE-CS, 1995.

[4] S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. *SIGMOD Record*, 26(1):65–74, 1997.

[5] K. T. Claypool, J. Jin, and E. A. Rundensteiner. SERF: schema evolution through an extensible, re-usable and flexible framework. In *Proceedings of the 1998 ACM 7th Int.*

*Conf. on Information and Knowledge Management (CIKM)*, pages 314–321, 1998.

[6] J. Clear, D. Dunn, B. Harvey, M. L. Heytens, P. Lohman, A. Mehta, M. Melton, L. Rohrberg, A. Savasere, and R. M. W. ans Melody Xu. NonStop SQL/MX Primitives for Knowledge Discovery. In *Proc. 5th ACM SIGKDD Int. Conference on Knowledge Discovery and Data Mining 1999, San Diego, CA USA*, pages 425–429, 1999.

[7] W. Cohen. Integration of Heterogeneous Databases Without Common Domains Using Queries Based on Textual Similarity. In L. Haas and A. Tiwary, editors, *SIGMOD 1998, Proc. ACM SIGMOD Int. Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 201–212. ACM Press, 1998.

[8] CRISP-DM Consortium. *CRISP 1.0 Process and User Guide*, 1998. http://www.crisp-dm.org/.

[9] U. Fayyad and K. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proc. 13th Int. Joint Conf. on Artificial Intelligence, Chambery, France*, pages 1022–1027. Morgan Kaufmann, 1993.

[10] H. Galhardas, D. Florescu, D. Shasha, and E. Simon. Ajax: An extensible data cleaning tool. In W. Chen, J. F. Naughton, and P. A. Bernstein, editors, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, volume 29, page 590. ACM, 2000.

[11] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *Journal of Intelligent Information Systems*, 8(2):117–132, Mar./Apr. 1997.

[12] J. Grant, W. Litwin, N. Roussopoulos, and T. Sellis. Query Languages for Relational Multidatabases. *VLDB Journal*, 2(2):153–171, 1993.

[13] J. Han and Y. Fu. Dynamic Generation and Refinement of Concept Hierarchies for Knowledge Discovery in Databases. In *AAAI'94 Workshop on Knowledge Discovery in Databases (KDD'94), Seattle, WA*, pages 157–168, 1994.

[14] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, 2000.

[15] M. Hernandez and S. Stolfo. Real-world data is dirty: Data Cleansing and the Merge/Purge problem. *Journal of Data Mining and Knowledge Discovery*, 2(1):9–37, 1998.

[16] W. Kelley, S. Gala, W. Kim, T. Reyes, and B. Graham. Schema Architecture of the UniSQL/M Multidatabase System. In W. Kim, editor, *Modern Database Systems*, chapter 30, pages 621–648. ACM Press, New York, NJ, 1995.

[17] L. Lakshmanan, F. Sadri, and I. Subramanian. SchemaSQL - A Language for Interoperability in Relational Multi-Database Systems. In T. M. Vijayaraman, A. Buchmann, C. Mohan, and N. Sarda, editors, *VLDB'96, Proc. of 22th Int. Conf. on Very Large Data Bases, 1996, Mumbai (Bombay), India*, pages 239–250. Morgan Kaufmann, 1996.

[18] A. Levy, A. Rajaraman, and J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In T. Vijayaraman, A. Buchmann, C. Mohan, and N. Sarda, editors, *VLDB'96, Proc. of 22th Int. Conf. on Very Large Data Bases, 1996, Mumbai (Bombay), India*, pages 251–262. Morgan Kaufmann, 1996.

[19] E.-P. Lim and R. Chiang. A Global Object Model for Accommodating Instance Heterogeneities. In T. W. Ling, S. Ram, and M.-L. Lee, editors, *Conceptual Modeling - ER '98, 17th International Conference on Conceptual Modeling, Singapore, November 16-19, 1998, Proceedings*, volume 1507 of *Lecture Notes in Computer Science*, pages 435–448. Springer, 1998.

[20] E.-P. Lim, J. Srivastava, and S. Shekhar. Resolving Attribute Incompatibility in Database Integration: An Evidential Reasoning Approach. In *Proc. of the 10th IEEE Int. Conf. on Data Engineering, ICDE'94, Houston, Texas, USA, 14–18 February 1994*, pages 154–163, Los Alamitos, CA, 1994. IEEE Computer Society Press.

[21] H. Lu, W. Fan, C. Goh, S. Madnick, and D. Cheung. Discovering and Reconciling Semantic Conflicts: A Data Mining Perspective. In *Proceddings of the 7th IFIP 2.6 Working Conference on Data Semantics (DS-7), Leysin, Switzerland*, 1997.

[22] F. Olken. *Random Sampling from Databases*. PhD thesis, UC Berkeley, April 1993.

[23] Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. Object Fusion in Mediator Systems. In T. Vijayaraman, A. Buchmann, C. Mohan, and N. Sarda, editors, *VLDB'96, Proc. of 22th Int. Conf. on Very Large Data Bases, 1996, Mumbai (Bombay), India*, pages 413–424. Morgan Kaufmann, 1996.

[24] K. Sattler and G. Saake. Supporting Information Fusion with Federated Database Technologies. In S. Conrad, W. Hasselbring, and G. Saake, editors, *Proc. 2nd Int. Workshop on Engineering Federated Information Systems, EFIS'99, Kühlungsborn, Germany, May 5–7, 1999*, pages 179–184. infix-Verlag, Sankt Augustin, 1999.

[25] K.-U. Sattler, S. Conrad, and G. Saake. Adding Conflict Resolution Features to a Query Language for Database Federations. *Australian Journal of Information Systems*, 8(1):116–125, 2000.

[26] E. Sciore, M. Siegel, and A. Rosenthal. Using Semantic Values to Facilitate Interoperability Among Heterogeneous Information Systems. *ACM Transactions on Database Systems*, 19(2):254–290, June 1994.

[27] A. Tomasic, R. Amouroux, P. Bonnet, O. Kapitskaia, H. Naacke, and L. Raschid. The distributed information search component (disco) and the world wide web. In J. Peckham, editor, *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pages 546–548. ACM Press, 1997.

[28] S. Venkataraman and T. Zhang. Heterogeneous Database Query Optimization in DB2 Universal DataJoiner. In A. Gupta, O. Shmueli, and J. Widom, editors, *VLDB'98, Proc. of 24rd Int. Conf. on Very Large Data Bases, 1998, New York City, New York, USA*, pages 685–689. Morgan Kaufmann, 1998.

[29] J. Vitter. An Efficient Algorithm for Sequential Random Sampling. *ACM Transactions on Mathematical Software*, 13(1):58–67, March 1987.

[30] H. Wang and C. Zaniolo. User-Defined Aggregates for Datamining. In *1999 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 1999.