DATA CONVERSION, AGGREGATION AND DEDUCTION FOR ADVANCED RETRIEVAL FROM HETEROGENEOUS FACT DATABASES

Kalervo Järvelin+ and Timo Niemi#
+Dept. of Information Studies
#Dept. of Computer Science
University of Tampere
P.O.Box 607 SF-33101 TAMPERE, Finland

Abstract : Modern distributed fact databases are heterogeneous and autonomous. Their heterogeneity is due to many reasons, including varying data models, data structures, attribute naming conventions, units of measurement or naming of data values, composition of data as attributes, technical representation of data, abstraction levels of data, etc. Database autonomity means that the database users have hardly any means for reducing such heterogeneity. Present information retrieval (IR) systems either provide no support for overcoming such heterogeneity or their support is insufficient and difficult to utilize. In this paper we offer integrated and powerful data conversion, aggregation and deductive techniques for advanced IR in such environments. These techniques allow the users to overcome data inconsistency due to units of measurement or naming of data values, composition of data as attributes, abstraction levels of data, and difficulties related to deductive use of hierarchically classified data. In complex situations, all these inconsistencies appears together. Therefore we also show how these techniques are integrated into a powerful query language which has been implemented in Prolog in a workstation environment.

1. INTRODUCTION

The characteristics of modern distributed fact database environments make their utilization by end-users and information specialists difficult. There are thousands of fact databases available through many diverse databanks and networks. It is hard to keep track on what is information available, where and how. The databases in such environments are heterogeneous and autonomous. Their heterogeneity may be due to many reasons, e.g. data models, data structures, attribute naming conventions, units of measurement or naming of data values, composition of data as attributes, technical representation of data, abstraction levels of data, etc. Therefore it is difficult to bring data from different sources together. Often the data item values are based on classifications which the users must consult in separate files. Autonomity means that there is no global control on database characteristics and the users in particular cannot exercise such control. [1] [2] [5] [18]

The literature uses many terms, e.g. numeric, statistical, factual, non-bibliographic databases or databanks to refer to public access databases which provide structured data [1] [5]. In this paper we use the term fact database. We assume for simplicity that the data are stored in (heterogeneous) relational databases. We assume that the reader is familiar with the relational model (RDM) and the relational algebra (RA).

The characteristics of the distributed fact database environments have made it necessary to

develop tools on all levels to aid database access. Studies in the area concern data structure conversion (e.g. [15]), communication with remote databases, deductive capabilities (e.g. [4] [10]), integration of relational databases and statistical software (e.g. [9]), and object-oriented systems aiming especially at hiding technical details of databases from remote users (e.g. [2] [14]).

We introduce integrated and powerful data retrieval, conversion, aggregation and deductive operations which are necessary for advanced information retrieval (IR) in fact database environments. They allow the users to overcome data inconsistency due to units of measurement or naming of data item values, composition of data as attributes, data abstraction levels, and difficulties in the utilization of classifications. These problems are frequent in the use of fact databases.

We mean by *data conversion* the conversion of data item *values* from one domain (i.e. a set of possible values) to another rather than structural conversion (e.g. [15]). This may also involve recomposition of data as attributes, e.g. replacing day, month and year -values by a unary date-value. We present a *generalized and powerful data conversion operation* as an RA operation which supports unit-of-measurement transparency and requires minimal effort from the user. Although data conversion has been considered earlier as a tool for heterogeneous databases (e.g. [2]) and is available to a limited degree in some statistical databases [5], our operation is original in its generality and ease-of-use, and because it is defined as an optimizable relational operation.

Data aggregation means raising the abstraction level of data, i.e. classifying source data in some defined classes on the basis of some attribute values and computing means, sums, minimums etc. on some other attributes in these classes. The resulting data are on a new abstraction level. Simple, single level data aggregation is provided in conventional query languages (e.g. SQL, QBE [16]) and - with limited class definition possibilities - in fact database query interfaces (e.g. [18]). However, single level aggregation is insufficient. In particular, *multiple aggregation levels* and *classification attribute reclassification to hierarchical superclasses during aggregation* are vital to IR. We present a novel *powerful multi-level data aggregation operation* as an RA operation which supports aggregation level transparency and requires minimal effort from the user.

Much effort has been devoted in database research to developing *deductive* capabilities in data management (e.g. [10] [14] [17]). In practice the computation of transitive relationships among entities is the most important case of deriving new information from explicitly stored information. In this paper we introduce a novel set of deductive operations for processing transitive relationships.

We demonstrate the use of these operations in deductions based on classifications for data item values. Such classifications (of e.g. chemical compounds) are usual in IR. Conventional query languages and numeric database query interfaces provide insufficient support in the use of classifications (e.g. only a lookup possibility in a separate classification file [18]). Traditional IR systems (e.g. Medline) provide only a hierarchical expansion of query terms. Moreover, deductive database systems require that users specify deductions by Horn clauses which makes end-user access very difficult [13]. *Our deductive operations are integrated seamlessly with other query operations, are much easier to use than Horn clauses and enable powerful utilization of classifications.*

Encapsulation of technical details of distributed databases through e.g. object oriented techniques is very important for simplifying IR (e.g. [2]). For example, users of relational databases must know relation and attribute semantics, join conditions and their semantics in detail in order to use them properly. In distributed environments there are many relations changing both in structure and content, and users thus have difficulties. In this paper we present entity types and an entity data retrieval operation as a *high-level interface* to relational databases which combines the strengths of object-oriented database systems (e.g. [14]) and universal relation systems (e.g. [17]) *for simple and semantically accurate retrieval.* However, our entity types and instances are not limited to RDBs [6] [12].

Summing up, features of fact database user interfaces and conventional query languages do not meet user requirements : higher level interfaces providing better transparency and greater expressive power are needed. The operations presented in this paper form a powerful, flexible and coherent operation-oriented query language for heterogeneous database access and provide essential support in overcoming the limitations of present systems. We shall focus on the operations and the expressive power of our query language and bypass many details of user interfaces, data communication and technical implementation.

In Chapter 2 we present our sample environment and introduce the representation of classifications as binary relations and the entity types based on relational databases. In Chapter 3 we

introduce our operations for processing transitive relationships, data retrieval, conversion and aggregation. Chapter 4 exemplifies our integrated query language in solving complex queries. Chapter 5 contains discussion and conclusions.

2. SAMPLE WEATHER AND POLLUTION DATABASES

Our imaginary database environment provides data on weather and air pollution in Great Britain and Germany. The data are gathered from sensor stations located in various towns, cities and their surroundings. Each town or city with its surroundings is called a location. The environment consists of two databases (London, Frankfurt) which are structured differently. The environment uses classifications for the attribute values. We shall first introduce the classifications, followed by the entity based query interface. In the final section we discuss the inconsistency of the environment.

2.1. Sample Classifications and Thesauri

Our sample environment contains several classifications given in the Appendix. The classification **air-pollutant** classifies pollutants to pollutant families and pollutant types, and the classification **health-hazard** classifies pollutants to health-hazard types (e.g. carcinogenes). The geographical classifications **great-britain** and **germany** classify locations to sub-regions, regions and countries. The sensor stations are classified to locations by the classifications **gb-station-location** (for Great Britain) and **ger-station-location** (for Germany). The sensor stations are also classified to station types (e.g. urban) by the classifications **gb-station-type** and **ger-station-type**. Classifications with the prefix **gb-** (**ger-**) are specific to the London (Frankfurt) database.

Classifications are represented as collections of binary relations. These provide a natural, simple and general representation for deduction (e.g. [4]). Transitive class relationships (e.g. finding recursively all subclasses of a class) are represented easily by binary relations which store explicitly the immediate class relationships.

Each classification is represented so that each pair of immediate hierarchical levels (i.e. each level of subdivision) forms a binary relation. Thus the binary relations for the classification **air-pollutants** are as outlined in Fig. 2.1. Each classification level forms its own domain of values. The root domain for **air-pollutants** has the name *air-pollutant* and contains the single value air-pollutant. The first level domain is *poll-type* containing the values {solid-pollutant, gas-pollutant}. The second level domain is *poll-family* and the leaf domain is *pollutant*. The first binary relation for **air-pollutants** is **pollutant-types** which is a relation in the Cartesian product *air-pollutant* × *poll-type*. Analogously, the other binary relations are **pollutant-families** and **pollutants** which are relations in *poll-type* × *poll-family* and *poll-family* × *pollutant*, respectively.

The Appendix contains the names of the domains related to the levels of each classification as well as names and signatures of the associated binary relations. It is easy to translate the classification levels to respective domains and the hierarchical divisions to binary relations. They are used below. The binary relations are called briefly ECRs (Extensional Classification Relation). They must not be mixed with regular relations.

pollutant-types	
air-pollutant	solid-pollutant
air-pollutant	gas-pollutant

pollutant-families	
solid-pollutant	soot
solid-pollutant	dust
gas-pollutant	COx
gas-pollutant	NOx
...	...

pollutants	
soot	diesel-soot
soot	coal-soot
NOx	NO
NOx	NO2
SOx	SO2
CFC	C2H5F
CFC	C3HCl4F
HxCxClx	C14H9Cl5
...	...

Fig. 2.1. Binary relations for **air-pollutants**

Our ECR organization has the advantage that the treatment of classifications can be focused to exactly those consequtive classification levels which are of interest. For example, a classification overview is obtained by considering hierarchical relationships only among **pollutant-types** and **pollutant-families**. Our approach supports multiple distinct classifications of the same basic entities. For example, the classifications **air-pollutant** and **health-hazard** classify the same basic pollutants on the basis of a different aspect. These features are necessary for IR (cf. the use of thesauri in traditional IR) but not provided by current deductive databases which define transitive relationships in a single large binary relation.

When the subclasses of each single classification are disjoint at all levels, the binary relations representing them can be interpreted as functions from the subclass domain to superclass domain. That is, given a subclass, the superclass can be identified. This feature allows very powerful data conversion and aggregation.

The classifications are augmented by thesauri, which give for each relevant term its domain, class name, definition, synonyms, and related terms (broader and narrower terms are given by the classifications), etc. The domain description relates each term (and class) to a particular domain of values. The attributes of the entity types and relations (see below) are declared over given domains. It is thus possible to find for any attribute value its synonyms and other information in the thesaurus. Contrary to [3], we associate the thesauri with domains rather than attributes, because the same domain may be shared by several attributes. The thesauri are represented to the users as entity types in the way described in Section 2.2. Here we bypass them because their content and use are quite obvious in IR.

2.2. Sample Entity Types In our framework, real world entities are represented by entity types with their instances which have attributes. An entity type gives an entity type name (e.g. person), and a set of entity attribute names (e.g. name, age) with their domains (e.g. string, integer). This notion corresponds, on a higher level, to the notion of relation schema in the RDM. An entity instance is a set of entity tuples of a given entity type (cf. the instance of a relation). An entity tuple is a tuple of values (e.g. <smith, 34>). Entity types are structured and named from the user's point of view - independent of the stored database relations. Their implementation is encapsulated.

Database users see only the entity level of the database. Our framework combines the strengths of the universal relation (e.g. [17]) and object-oriented approaches (e.g. [14]) into an entity-based query interface and provides enhanced data independence, accurate query semantics, and high-level query optimization [6] [13].

The entity types of our sample environment are given in Figs. 2.2 and 2.3. Each entity type has an entity type name (e.g. Daily-Weather-Observation) and a set of entity attribute names together with their domains (e.g. DailyMinTemp, °F ; Station, gb-stat-name ; etc.). All entity attributes are not shown as denoted by '...'. This is the part of entity types the users may see and use in their queries. The queries are thus build on entity type names and entity attribute names.

Figs. 2.2 and 2.3 also list the relational attributes (*rdb-attributes*) corresponding to entity attributes. For each of the latter, there may be several corresponding relational attributes, e.g. in the entity type Daily-Weather-Observation, the relational attributes TStat and WStat correspond to the entity attribute Station. The values of an entity attribute can be retrieved from any corresponding relational attribute depending on the combination of entity attributes in the whole request. [6] [13]

Entity-type : Daily-Weather-Observation		
attribute name	**domain**	**rdb-attributes**
DailyMinTemp	°F	DMinTmp
DailyMaxTemp	°F	DMaxTmp
DailyAverTemp	°F	DAvTmp
WindSpeed	mph	WSpeed
Pressure	mmbar	Press
...	...	...
Station	gb-stat-name	TStat, WStat
Day	day-int	TDay, WDay
Month	month-ch	TMonth, WMonth
Year	year-int	TYear, WYear

Entity-type : Hourly-Pollution-ObservationI		
attribute name	**domain**	**rdb-attributes**
Pollutant	pollutant	Ptant
Content	Oz-per-ft^3	PMesment
...	...	...
Hour	hour-int	PTime
Station	gb-stat-name	PLoc, GLoc
Day	day-int	PDay, GDay
Month	month-ch	PMonth, GMonth
Year	year-int	PYear, GYear

Fig. 2.2. Entity types in the London database

Entity-type : Hourly-Weather-Observation		
attribute name	**domain**	**rdb-attributes**
Temperature	°C	Temp
WindSpeed	mps	WSpeed
Pressure	MPascal	Press
...	...	...
Station	ger-stat-name	TStat, RStat
Hour	hour-int	THour, RHour
Date	date-int	TDate, RDate

Entity-type : Hourly-Pollution-ObservationII		
attribute name	**domain**	**rdb-attributes**
Pollutant	pollutant	Ptant
Content	gr-per-m^3	PMesment
...	...	...
Hour	hour-int	GHour
Station	ger-stat-name	GStat
Date	date-int	GDate

Fig. 2.3. Entity types in the Frankfurt database

2.3. Inconsistency of the Sample Environment Our sample fact database environment is clearly very inconsistent because :

- The two databases do *not* give explicitly *the same attributes* (e.g. London provides Dai-

lyAverTemp, DailyMinTemp, etc. while Frankfurt provides only Temperature).
- Corresponding information is given by *different numbers of attributes* (e.g. Date in Frankfurt vs. Day, Month, Year in London).
- Semantically and structurally equivalent attributes have *different units* (e.g. the attributes WindSpeed and Pressure have ISO units in Frankfurt and anglo-american units in London).
- Semantically strongly related data are at *different aggregation levels* (e.g. hourly weather data in Frankfurt vs. daily weather data in London).

Thus the data from the two databases cannot be put together without serious errors. Such inconsistencies are frequent among fact databases and form an important problem in IR. Due to database autonomity it is hardly possible to standardize them into directly compatible forms. Tools are thus needed for data standardization during retrieval.

In addition, it is difficult to obtain summary data based on the classifications for e.g. chemical compounds or geographical areas. The data aggregation tools in ordinary query languages cannot utilize external classifications - they support only aggregation on explicit values in the data.

3. DATA RETRIEVAL, CONVERSION, AGGREGATION AND DEDUCTION OPERATIONS

3.1. Relational Algebra Relational languages (e.g. SQL) as such are at a too low level for basic database access. The problems of a relational language are essentially reduced when it is used only for combining subquery results, and the subqueries are expressed in terms of higher level operations. In such a case there are only a few (intermediate) relations to be considered and the user knows their semantics. We use RA for combining subquery results.

Usual RA operations are allowed in our query language. The operations can be nested in more complex expressions in the usual way. Below we allow also the entity data retrieval operation and the data conversion and aggregation operations as subexpressions in RA expressions.

3.2. The Operation-Oriented Query Language for Classifications Our operation-oriented query language for classifications provides several types of ECR-based operations : *non-transitive operations, set-oriented* and *path-oriented transitive operations* as well as an *aggregation operation for paths* [13]. Here we shall introduce only some set-oriented operations. Formal and complete treatment is in [11] [13].

The set operations `set_intersection(set1, set2)`, `set_union(set1, set2)`, and `set_difference(set1, set2)` can be applied on sets of classes. The basic transitive set-oriented operations are the operations `successors(Class, Scope)` and `predecessors(Class, Scope)` which give all subclasses or superclasses of a class (the argument `Class`) in the given collection of ECRs (the argument `Scope`). The subclasses (superclasses) of a class are its all subclasses (superclasses) at all lower (higher) levels of hierarchy within `Scope`.

The two operations `union_of_successors(Class-set, Scope)` and `union_of_predecessors(Class-set, Scope)` compute all distinct subclasses or superclasses of given classes (the argument `Class-set`) in the collection of ECRs (the argument `Scope`). The operations `intersection_of_successors(Class-set, Scope)` and `intersection_of_predecessors(Class-set, Scope)` are used to find the common subclasses or superclasses of classes (the argument `Class-set`) in the given ECRs (the `Scope`). Analogous operations exist for the difference of successors and predecessors (bypassed here).

The operation expressions are structured with the keywords `class`, `set` and `scope`. Nested expressions are allowed. For example, the expression below finds all air-pollutant classes that are superclasses both to at least one carcinogene and at least one poison :

```
set_intersection(
  union_of_predecessors(
    set: successors(class: carcinogenes
           scope: haz-compounds)
    scope: pollutant-types, pollutant-
         families, pollutants)
  union_of_predecessors(
    set: successors(class: poison
           scope:  haz-compounds)
    scope: pollutant-types, pollutant-
         families, pollutants))
```

and gives {HxCxClx, gas-pollutant,air-pollutant}.

Our deductive operations provide a powerful yet simple foundation for the treatment of classifications. The standard interface (recursive Horn-clauses) for defining deductive queries in data management prevents end users from using deductive retrieval [13]. The expressive power is far better than plain hierarchical expansion of traditional IR systems. High-level deductive operations are necessary for practical advanced IR.

3.3. The Entity Data Retrieval Operation The entity data retrieval operation has three arguments which resemble those in conventional query languages (e.g. QUEL [16]) : the *retrieve specification,* the *variable binding* and the *predicate.* However, here they define processing of entity types and instances instead of relations. This simplifies retrieval considerably because the user does not need to know about the (possibly many) relations storing the data. [6] [13]

The variable binding associates *entity tuple variables* with the entity types relevant to a query. It is expressed as a `range`-clause, e.g. `range w IS-A daily-weather-observation` states that the variable `w` varies over the tuples of the instance of the entity type `daily-weather-observation`.

The retrieve specification lists the entity attributes to be output, prefixed by their entity tuple variables. For example, if one wants to retrieve the min and max temperatures associated with the entity tuple variable `w`, the expression is `retrieve w.DailyMinTemp, w.DailyMaxTemp`.

The predicate states the conditions which must hold when data are output. For example, if output is required when the tuple associated with the tuple variable `w` has a Pressure -value exceeding 1040, WindSpeed less than 1 mph and Year and Month -values 1991 and Jan, respectively, the expression is `where w.Pressure > 1040 and w.WindSpeed < 1 and w.year = 1991 and w.month = jan`. Usual logical operators and parentheses are allowed. The whole query for the min and max temperatures for high-pressure calm days in January 1991 is as follows :

```
retrieve w.DailyMinTemp, w.DailyMaxTemp
range w IS-A daily-weather-observation
where w.Pressure > 1040 and w.WindSpeed
< 1 and w.Year = 1991 and w.Month = jan.
```

The corresponding SQL-expression depends on the number of relations storing the data. The user should know their join semantics

When there occurs a need to refer to the resulting attributes of the operation, aliases (in the SQL-style, e.g. [16]) can be given in the `retrieve` -clause. For example, `retrieve w.DailyMinTemp Temp` associates the alias `Temp` with the result attribute `w.DailyMinTemp`.

One entity data retrieval operation can process any number of entity types. Each entity type is associated with its own distinct entity type variable in the `range` -clause.

3.4. The Data Conversion Operation The data conversion operation converts attributes to new domains, e.g. from inches to meters. It has the following important generalized properties :

- Any number of attributes or combinations of attributes may be converted at once. Attribute combinations consist of two or more attributes (e.g. day, month and year) which together have a distinct meaning (e.g. date).
- One operand attribute may be converted into an attribute combination or a different attribute in the result and vice versa.
- The user only needs to name the operand and result attributes and the result domain. The operation itself checks the derivability of the conversion and forms the conversion function.
- The conversion functions may convert the unit of measurement as well as classify the attribute values. In the former case the conversion is an injection : one value in e.g. inches is converted into exactly one value in meters. In the latter case the conversion maps many domain values to the same range value, e.g. dates to year-months which means that all dates from 010191 to 310191 are mapped to 9101 for Jan, 1991.
- The conversion functions are defined in the database but all need not be defined explicitly. For example, if the conversion functions f from miles to feet, and g from feet to meters are defined, any value x in miles can be converted to a meter value by the compound function g(f(x)). The operation compounds all implicit conversion functions automatically.

The expression below converts the pollutant content in the London database from ounces-in-cubic-feet (oz-in-ft^3) to the new domain grams-in-cubic-meter (gr-in-m^3, injective) of the result attribute `Poll-cont` and classifies pollutants to the new domain pollutant-family (not injective) of the result attribute `P-group`. It also converts the three attributes day (e.g. 28), month (e.g. feb) and year (e.g. 91) to one date value (e.g. 910228).

```
convert  hourly-pollution-observationI
columns  (Content, gr-in-m3, Poll-cont),
         (Pollutant, poll-family, P-group),
         (Day Month Year, date-int, Date)
```

The conversion functions are defined in the database by conversion descriptions which give for a domain all domains into which the former can be converted, and their conversion functions. For example, the conversion description (a) in Fig. 3.1 means that the domain gr-in-m^3 (for grams-in-m^3) can be converted into the domain oz-

in-ft^3 (for ounces-in-ft^3) by the function f(x). The conversion description (b) shows a conversion description for the classifications. The inverse relation of the relation pollutants is the function g: pollutant → poll-family. For example, g(NO2) = NOx (see Appendix). The conversion descriptions are utilized also by the aggregation operation.

(a) **cd**(gr-per-m3 , {(oz-per-ft3, f(x) = x × 0.0009996, injection)})
(b) **cd**(pollutant, {(poll-family, g(x), function)})

Fig. 3.1. Sample conversion descriptions

3.5. The Data Aggregation Operation The data aggregation operation has been developed for aggregating data to different abstraction levels, e.g. from hourly to daily level. It provides new properties which expand aggregation possibilities :

- The data can be aggregated in *multiple consequtive aggregation levels* each of which is defined by several classification attributes.
- The values of the *classification attributes can be reclassified to their hierarchical superclasses* at any level of hierarchy or converted to any other domain allowed by the conversion functions.

Multiple aggregation levels and classification attribute reclassification are essential for advanced IR. Classifications of e.g. chemical compounds and time intervals are needed, and higher level sums, averages etc. of lower level minimums, averages, etc. are often requested but not supported well enough by conventional query languages nor by commercial fact database software. Conventional query languages, e.g. SQL, provide single level aggregation but no possibility for reclassifying the class attribute values. The query interfaces of fact databases seem to support only single level aggregation on a restricted set of classification attributes and do not seem to provide choices on the aggregation way (sum, count, ...) [18]. They support limited classification attribute reclassification (e.g. monthly vs. yearly data) [7] [18].

Other properties of the aggregation operation include the following :

- The aggregation ways : min, max, sum, count and average are allowed. Weighted aggregation and standard deviation are easy extensions.
- Any number of operand attributes may be aggregated at once. One attribute may be aggregated in several different ways.
- The user names the operand attributes to be aggregated and gives stepwise the aggregation ways and result attribute names on each aggregation level, e.g. `(Content aver DailyAverCont, max MonthMaxDailyCont)` defines that first the average for the attribute `content` is calculated into the attribute `DailyAverCont` and then the maximum of the latter is computed into `MonthMaxDailyCont`. Such definitions are called *attribute aggregation definitions.*
- For each aggregation level, the user gives an *aggregation class former* which is a set of one or more *facets* consisting of three components : the *classification attribute name,* the *name of the domain* into which the classification attribute is reclassified, and the *result attribute name.* For example, `(Station gb_stat_name Station, Pollutant poll_family PollGroup)` is an aggregation class former which classifies all data for each sensor station and pollutant family together. The last facet `Pollutant poll_family PollGroup` causes that individual pollutants are classified into pollutant families. The operation checks conversion derivability and forms the conversion function as explained above. If no reclassification is needed then the operand and result attribute names are the same.
- The aggregated attributes appear in the result. In addition, the user can choose any operand attributes and any classification attributes of the last aggregation level into the result.
- The user can restrict the result by a conjunctive predicate which may refer to any operand attributes, aggregated and classification attributes.

For example, the expression below gives year-monthly max and min of average daily contents of pollutant families NOx and SOx:

```
aggregate(retrieve t.Pollutant Po,
     t.Content Cont, t.Day Day, t.Month
     Mo, t.Year Yr
   range  t IS-A daily-pollution-
       observation
   where  t.Station = london1)
columns (Cont aver DAvC1, max MMxDCont),
     (Cont aver DAvC2, min MMnDCont)
class  (Day day_int Day, Mo month_ch Mo,
   Yr year_int Yr, Po poll_family PoFa)
class  (Mo month_ch Mo, Yr year_int Yr,
   PoFa poll_family PoFa)
select MMxDCont, MMnDCont, Mo, Yr, PoFa
where  PoFa = one_of(SOx, NOx)
```

The keyword `columns` precedes the attribute aggregation definitions, the keyword `class` denotes each aggregation class former, the keyword `select` denotes the projected attributes, and the keyword `where` the predicate. The attribute `Cont` is the only aggregation attribute and `DAvC1`, `MMxDCont`, `DAvC2`, and `MMnDCont` the aggrega-

tion result attributes through the two aggregation levels. The aggregation ways are min, max and aver. In the first aggregation class former, `Day`, `Mo`, `Yr` and `Po` are the classification attributes, `day_int`, `month_ch`, `year_int` and `poll-family` their domains (for reclassification if the domains differ), and `Day`, `Mo`, `Yr` and `PoFa` the corresponding classification result attributes. In each attribute aggregation definition (say `(Cont aver DAvC1, max MMxDCont)`), the first pair of an aggregation way and result attribute (e.g. `aver DAvC1`) always relates to the first aggregation class former, the second pair to the second etc.

4. INTEGRATED QUERIES

The entity data retrieval, aggregation, and conversion operations are defined at the RA level and can be mixed freely with each other and RA in nested expressions. Deductive operations can also be integrated with them via the predicate of type <attribute-name> one-of(<value-set>) which requires that the attribute has some value in the value-set. We allow that the value-set is formed by a deductive expression [13].

```
union(
  (convert
    (aggregate
      (retrieve p1.Pollutant Po1, p1.Content Co1, p1.Hour Hr1, p1.Station
            St1, p1.Day Day1, p1.Month Mo1, p1.Year Yr1
       range p1 IS-A Hourly_Pollution_observationI
       where St1 one_of(union_of_successors(set: metropolitan, urban
                     scope: gb-typed-stations)) and
            Po1 one_of(union_of_successors(set: gas-pollutant
                     scope: pollutant-families, pollutants)))
     columns (Co1 sum DCo1, max MxDCo1)
     class   (St1 gb-location City1, Day1 day-int Day1, Mo1 month-ch Mo1, Yr1
            year-int Yr1, Po1 poll-family PoFam1)
     class   (City1 gb-location City1, Mo1 month-ch Mo1, Yr1 year-int Yr1, PoFam1
            poll-family PoFam1)
     select  MxDCo1, PoFam1, City1, Mo1, Yr1
     where   true)
    columns  (MxDCo1, gr-per-m3, MxDCo1gr), (Mo1 Yr1, year-month-int, YrMo1))
  (aggregate
    (retrieve p2.Pollutant po2, p2.Content Co2, p2.Hour Hr2, p2.Station
          St2, p2.Date Da2
     range p2 IS-A Hourly_Pollution_observationII
     where St2 one_of(union_of_successors(set: metropolitan, urban
                     scope: gb-typed-stations)) and
          po2 one_of(union_of_successors(set: gas-pollutant
                     scope: pollutant-families, pollutants)))
    columns (Co2 sum DCo2, max MxDCo2)
    class  (St2 ger-location City2, Da2 date-int Da2, Po1 poll-family PoFam1)
    class  (City2 ger-location City2, Da2 year-month-int YrMo2, PoFam2 poll-
          family PoFam2)
    select MxDCo2, PoFam2, City2, YrMo2
    where  true))
```

The query above demonstrates the queries expressible in our integrated query language. It gives monthly maximums of daily averages of gas-pollutant content for gas-pollutant families in cities from the two databases in a consistent form. Remote database access is not shown.

5. DISCUSSION AND CONCLUSIONS

Distributed heterogeneous databases are autonomous and thus users must adapt themselves to the databases. Here they need well-defined high-level tools. We have demonstrated that our deductive, data retrieval, conversion and aggregation operations provide firm tools for overcoming data inconsistency. The operations provide the users, among others, with rich expressive power, a homogeneous way of query formulation (also in complex cases), means for achieving unit-of-measurement and abstraction level transparency, a higher level knowledge representation than relations, and modular query formulation. The op-

erations support user adaption to, and coordinated data access in, heterogeneous environments.

The sample query shows that users may formulate queries which go up and down in classifications, retrieve data for the classes found and filter them by some conditions, aggregate them and pick some resulting aggregations to go to other classifications, etc. Obviously, very compact expressions cannot be formed for complex needs without restricting the user's expressive power. The sample expression for complex needs is not simple for casual end-users. Yet our operations simplify query expression in heterogeneous database environments essentially and provide an expressive power which so far has been available only for users with advanced programming skills.

Our operations provide important new features for fact retrieval : The *deductive operations* integrate deductions based on classifications seamlessly with the other operations so that the classifications can be utilized effectively and flexibly in retrieval. The *entity data retrieval operation* simplifies retrieval by encapsulating the implementation of high-level entity types. The *data conversion operation* provides unit-of-measurement transparency. It supports very versatile conversion (including conversion of compound attributes), checks automatically the derivability of conversion requests and requires minimal information from the user. *The data aggregation operation* provides abstraction level transparency. It supports, among others, multiple aggregation levels and hierarchical reclassification of the classification attributes defining the aggregation levels.

We are working on hierarchical data structures which are accessed via entity types with the same convenience as relations and we are extending the entity type representation and management so that the needs for data conversion and aggregation can be detected, and necessary operation sequences synthesized, automatically in the construction of entity instances. All the operations presented have been implemented in Prolog in a workstation environment. Details of the operations are given in [6] [8] [11] [12] [13].

REFERENCES

[1] Chen, C. & Hernon, P. (Eds.), Numeric Databases. Norwood : Ablex, 1984.

[2] Connors, T. & Lyngbaek, P., Providing uniform access to heterogeneous information bases. In : Dittrich, K.R. (Ed.) Advances in Object-Oriented Database Systems. Lecture Notes in Computer Science, 334. Berlin: Springer, 1988 : pp. 162-173.

[3] Hoppe, K. & al., EXPRESS : An Experimental Interface for Factual Information Retrieval. In: J-L. Vidick (Ed.), Proc. 13th International Conf. on Research and Development in Information Retrieval, 1990. Bruxelles: ACM, 1990 : pp. 63-81.

[4] Jagadish, H.V. & Agrawal, R., A Study of Transitive Closure as a Recursion Mechanism. Proc. of the 1987 ACM SIGMOD Conf., San Francisco, 1987. New York: ACM Press, 1987, pp. 331-334.

[5] Järvelin, K., A Methodology for User Charge Estimation in Numeric Online Databanks. J. Inform. Sci., 14 (1) : 3-16 ; 1988.

[6] Järvelin, K. & Niemi, T. Entity-Based Query Construction for Relational Database Access. Tampere: U. of Tampere, Dept of Computer Sci., Report A-1990-6, 1990.

[7] Järvelin, K. & Niemi, T. Advanced Retrieval from Heterogeneous Fact Databases: Integration of Data Retrieval, Conversion, Aggregation and Deductive Techniques. Tampere: U. of Tampere, Dept of Information Studies, Res. Note RN-1991-1, 1991.

[8] Järvelin, K. and Niemi, T., General Data Conversion and Aggregation Operations : Definition and Integration with Data Retrieval and Deductive Techniques. Tampere: U. of Tampere, Dept of Computer Sci., Report Series A, 1991. (forthcoming)

[9] Leonard, M. & Snella, J.J. & Abdeljaoued, A., Farandole : A RDBMS for Satistical Data Analysis. In: De Antoni, F. & Lauro, N. & Rizzi, A. (Eds.) Proc. of 7th COMPSTAT Symposium in Computational Statistics, Rome 1986. Heidelberg: Physica-Verlag, 1986 : pp. 448-453.

[10] McLean, S. & Weise, C., Digress: A Deductive Interface to a Relational Database. J. Amer. Soc. Inform. Sci., 42(1),1991:49-63.

[11] Niemi, T. & Järvelin, K. Operation-Oriented Query Language Approach for Recursive Queries. Tampere: U. of Tampere, Dept of Computer Sci., Rep. A-1990-2, 1990.

[12] Niemi, T. & Järvelin, K. Prolog-Based Metarules for Relational Database Representation and Manipulation. IEEE Trans. on Software Eng., September 1991, to appear.

[13] Niemi, T. & Järvelin, K. Advanced Query Formulation in Deductive Databases. Inform. Proc. & Manag., 1991, to appear.

[14] Parsaye, K. & al., Intelligent Databases : Object-Oriented, Deductive Hypermedia Technologies. New York: Wiley, 1989.
[15] Ruschitzka, M. & Clevenger, J., Heterogeneous Data Translations Based on Environment Grammars. IEEE Trans. on Software Eng., 15(10), 1989 : 1236-1251.
[16] Ullman, J.D., Principles of Database and Knowledge Base Systems. Vol. I. Rockville: Computer Science Press, 1988.
[17] Ullman, J.D., Principles of Database and Knowledge Base Systems. Vol. II : The new technologies. Rockville: Computer Science Press, 1989.
[18] Wolski, K. & al. Numeroita näytöllä : Ulkomaisia numeerisia tietokantoja [Numbers on Screen : Foreign Numeric Databases]. Helsinki : Central Statistical Office of Finland, Handbooks 25, 1990.

APPENDIX

air-pollutant
solid-pollutant
soot: diesel-soot, coal-soot
dust: asbest-dust, ash-dust
gas-pollutant
COx: CO, CO2
NOx: NO, NO2
Ozone: O3
SOx: SO, SO2, SO3
CFC: C2H5F, C3HCl4F, C2Cl4F2, C2Cl2F4
HxCxClx: C2H5Cl, C2HCl5, C14H9Cl5

Domains : root = air-pollutant, first level = poll-type, second level = poll-family, leaf = pollutant
Binary Relations : pollutant-types: air-pollutant×poll-type, pollutant-families: poll-type×poll-family, pollutants: poll-family×pollutant

health-hazard
allergenes: ...
carcinogenes: diesel-soot, coal-soot, asbest-dust, ash-dust, C2HCl5
poisons: CO, O3, NO, C3HCl4F, C14H9Cl5

Domains : root = health-hazard, first level = hh-type, leaf = pollutant
Binary Relations : hh-types: health-hazard×hh-type, haz-compounds: hh-type×pollutant

gb-sensor-type
metropolitan:
london1, london2, sheffield1, ...
urban: london5, sheffield3, ...
countryside: cambridge1, ...

Domains : root = sensor-type, first level = station-type, leaf = gb-stat-name
Binary Relations : station-types: sensor-type×station-type, gb-typed-stations: station-type×gb-stat-name

ger-sensor-type
metropolitan:
stuttgart1, stuttgart2, münchen1, ...
urban: heidelberg2, ...
countryside: heidelberg3, ...

Domains : root = sensor-type, first level = station-type, leaf = ger-stat-name
Binary Relations : station-types: sensor-type×station-type, ger-typed-stations: station-type×ger-stat-name

gb-station-location
london: london1, london2, ...
sheffield: sheffield1, ...
...

Domains : root = gb-station-loc, first level = gb-location, leaf = gb-stat-name
Binary Relations : gb-s-locations: gb-station-loc×gb-location, gb-stat-names: gb-location×gb-stat-name

ger-station-location
stuttgart: stuttgart1, stuttgart2, ...
heidelberg: heidelberg1, ...
...

Domains : root = ger-station-loc, first level = ger-location, leaf = ger-stat-name
Binary Relations : ger-s-locations: ger-station-loc×ger-location, ger-stat-names: gb-location×ger-stat-name

great-britain
england
north-england: newcastle, ...
mid-england: sheffield, nottingham, ...
...
wales ...
...

Domains : root = great-britain, first level = gb-region, second level = gb-subregion, leaf level = gb-location
Binary Relations : gb-regions: great-britain×gb-region, gb-subregions: gb-region×gb-subregion, gb-locations: gb-subregion×gb-location

germany
baden-württemberg : stuttgart, heidelberg
schleswig-holstein: kiel, lübeck
...

Domains : root = germany, first level = ger-region, leaf level = ger-location
Binary Relations : ger-regions: germany×ger-region, ger-locations: ger-region×ger-location