Aggregates in Possibilistic Databases

Elke A. Rundensteiner and Lubomir Bic

Department of Information and Computer Science
University of California, Irvine

Abstract

Fuzzy set theory represents a uniform framework for extending the relational database model to handle imprecision of information found in the real world. None of the existing proposals for data models handling imprecision has dealt with queries involving aggregate operators. This paper presents a framework for handling aggregates in the context of imprecise information. Two kinds of aggregates, namely, scalar aggregates and aggregate functions, are being supported. We consider three cases: aggregates within approximate queries on precise data, aggregates within precisely specified queries on possibilistic data, and aggregates within vague queries on imprecise data. These extensions are based on fuzzy set-theoretical concepts such as the extension principle and the possibilistic expected value.

1 Introduction

It has been widely recognized that the uncertainty inherent in real world data has to be dealt with in database systems. Research addressing this problem has to a large extend been based on fuzzy set theory [11]. The two major objectives of these efforts are (1) enhancements to existing data models for representing incomplete and uncertain data, and (2) the development of new retrieval techniques for such data. The first issue addresses the limitation of conventional data models to allow attributes to take but one constant value from a domain [1, 8, 7]. The second problem is the inadequacy of current relational query languages (RQLs) to cope with different types of fuzzy representations of data. Most attempts to design an enhanced RQL, here called a FRQL [7, 12], are based on some form of the generally accepted relational algebra or relational calculus [2].

Proceedings of the Fifteenth International
Conference on Very Large Data Bases

Amsterdam, 1989

Database systems featuring only retrieval and update operations [2] are inadequate for many important applications [4]. Virtually all real-world problems need query capabilities involving aggregate and statistical functions [3]. This strongly suggests that the evaluation of aggregates has to be dealt with in the context of the extended relational models. However, most research on extending relational query languages has neglected this issue; the only notable exception is the work of Oezsoyoglu et al [5].

This paper investigates the incorporation of aggregate operators commonly found in traditional query languages like SQL or QUEL [3] into FRQLs. There are three general cases we consider. First, we handle approximate queries on a relational database containing precise information. An example of this situation is the query "What is the maximum salary of all *old* professors at UCI?" with the age and salary being precisely known and *old* being a fuzzy set defined on the Age attribute. The second case occurs when the query is exact but the information stored in the database is of possibilistic nature. An example of this case is the query "What is the maximum salary of professors at UCI?" where the salary is specified imprecisely in the database. The third case is obtained by combining the two approaches to handle queries with approximate restrictions against a database containing possibilistic specified data. The solutions proposed in this paper satisfy two principles, consistency and completeness.

The paper is structured in the following manner. First, we review the aggregate operators in the context of the relational data model (section 2). The basics of fuzzy set theory are presented in section 3. A possibilistic extension of the relational database model and the relational algebra is introduced in section 4, referred to as Possibilistic Relational Data Model (PRDM). Section 5 is devoted to our proposal for evaluating aggregates within the PRDM. Conclusions and future research are presented in the last section.

2 Aggregates in Relational Query Languages

In this section, we review how aggregates are handled within the classical relational data model [2, 4].

2.1 The Classical Relational Model

A relational data model consists of a set of *attribute names* A_i, a set of corresponding *domains* U_i, and a set of *relation schemas* R_i and relations r_i.

Definition 1 *A set of attributes $\{A_1, ..., A_n\}$ is called a* **relation schema** *$R(A_1, ..., A_n)$, or short R. Let U_i be the domain of the attribute A_i, with $1 \leq i \leq n$. A* **relation** *r on R is defined as a subset of the Cartesian product of the domains U_i, i.e., $r \subseteq U_1 \times ... \times U_n$. Each tuple $t_i \in r$ has the form $< a_{i1} \times ... \times a_{in} >$ with $a_{ij} \in U_j$. $t_i[A_j]$ denotes the value of tuple t_i on attribute A_j, i.e., $t_i[A_j] = a_{ij}$.*

Query languages for the relational database model (RQLs) are based on either the relational algebra or the relational calculus, both proposed by Codd [2]. In this paper, we use the relational algebra. We chose a notation similar to the one of QUEL [3] over a more formal mathematical one in order to make the queries more understandable. For a definition see [3].

2.2 Scalar Aggregates

Retrieval statements exclusively composed from retrieval operations are inadequate for most applications of database management systems [4]. Many real-world queries require the application of aggregate and statistical functions to database relations. Consequently, most commercially available systems provide a set of such aggregate operators [3]. In general, two types of aggregate operators are supported by RQLs; these are *scalar aggregates* and *aggregate functions* [3]. (For the latter see section 2.3).

Scalar aggregates take a set of tuples (a relation) as an argument and produce a single value as a result. The following describes their syntax and semantics.

Definition 2 *Let r be a relation on the relation schema $R(A_1, ..., A_n)$. The syntax for a scalar aggregate f on A_i of relation r is $f((A_i)(r))$. The semantics are defined by*

$$f((A_i)(r)) = y \text{ with } \quad y = f\{t[A_i] \mid t \in r\} \qquad (1)$$

A precise definition of the most common aggregate operators is provided next.

Definition 3 *Let the relation r defined on the relation schema R consist of n tuples, with $n \geq 0$. Let t be a tuple variable in r and A an attribute in R.*

1. $count((r.A)(r)) = n$
2. $sum((r.A)(r)) = \sum_{\forall t \in r} t[A]$ *if* $n > 0$
3. $min((r.A)(r)) = \min_{\forall t \in r} t[A]$ *if* $n > 0$
4. $max((r.A)(r)) = \max_{\forall t \in r} t[A]$ *if* $n > 0$
5. $avg((r.A)(r)) = \frac{1}{n} \sum_{\forall t \in r} t[A]$ *if* $n > 0$
6. $any((r.A)(r)) = sign(n)$ *where* $sign(n) = +1$ *if* $n > 0$ *and* $sign(n) = 0$ *if* $n = 0$.

2. to 5 are assumed to result in 0 whenever $n = 0$.

A scalar aggregate can be calculated independently from the rest of the query and then simply replaced by its value.

2.3 Aggregate Functions

Aggregate functions compute aggregations over one or more subsets of a relation. They first partition the relation on the values of some attribute and then compute the aggregation separately for each partition. The result of an aggregate function is a relation whose number of tuples equals the number of initial partitions. The result tuples consist of the attribute values on which the partition has been formed and the corresponding aggregate value for each partition. The syntax and underlying semantics [4] are shown below:

Definition 4 *Let r be a relation on $R(A_1, ..., A_n)$. The application of an aggregate function f on the attribute A_i is specified by f ((A_i)(r) BY A_j) with $1 \leq i,j \leq n$. The semantics of this are given by*

$$f((A_i)(r)BYA_j) = \{t[A_j] \circ y \mid (t \in r) \wedge$$

$$(y = f(\{t'[A_i] \mid (t' \in r) \wedge t'[A_j] = t[A_j]\}))\}. \qquad (2)$$

Definition 4 can easily be extended to partition on more than one attribute by replacing the attribute A_j by a set of attributes X with $X \subseteq R$. An example is given next.

Name	Salary	Position	Department
Tom	3500	Assistant	ComputerScience
Jack	4500	Full	ComputerScience
Julie	4000	Full	ComputerScience
Mary	2500	Associate	ComputerScience
Frank	3500	Associate	Engineering

Figure 1: The Prof relation

Example 1 *Given the relation Prof in figure 1. The query "What is the average salary of professors in Computer Science for each rank?" can be expressed by averageSal = avg((Prof.Sal) (SELECT (Prof WHERE Prof.Department = 'ComputerScience')) BY Prof.Position) The result of this is depicted in figure 2.*

Position	Avg-Sal
Assistant	3500
Associate	2500
Full	4250

Figure 2: The Average-Sal relation

As shown below, the evaluation of an aggregate function can be performed in several stages. This decomposition not only facilitates an understanding of the underlying semantics, but it will also serve as a vehicle for the remainder of the paper.

Definition 5 *Let A_i and A_j be attributes of the relation r. An aggregate function, $f((A_i)(r)$ BY $A_j)$, can be evaluated in the following manner:*

1. *First, the relational expression denoted by r is evaluated in the usual manner.*
2. *Then, the tuples are partitioned by the distinct values of the attribute A_j.*
3. *The aggregate operator f is applied to each partition P generated in step 2, i.e., the aggregate function f is replaced by a set of scalar aggregates of the form $f((A_i)(P))$ which are evaluated by definition 3.*
4. *The result of the aggregate evaluation for each partition is associated with the attribute value from A_j based on which the partition was performed.*

The partition in step 2 of definition 5 is straightforward for precise (crisp) values of A_j. The partition of the relation r on the attribute A_j corresponds to a function from the values a_i of the domain of A_j to a set of tuples taken from r defined by

$$P_r^{A_j}(a_i) = \{t \mid t \in r \wedge t[A_j] = a_i\}. \quad (3)$$

This can easily be extended to a partition on a set of attributes instead of just one attribute A_j.

3 Basic Concepts of Fuzzy Set and Possibility Theory

This section introduces the basic concepts of fuzzy set and possibility theory as proposed by Zadeh [11].

Definition 6 *Let U be a universe of discourse. F is a* **fuzzy subset** *of U, if there is a membership function $\mu_F : U \rightarrow [0,1]$, which associates with each element u $\in$ U a grade of membership $\mu_F(u)$ in F. A fuzzy set F is denoted by $F = \{\mu_F(u_1)/u_1, \mu_F(u_2)/u_2, ..., \mu_F(u_n)/u_n\}$ where $u_i \in U$ for $1 \leq i \leq n$.*

A close connection between fuzzy sets and possibility theory has been established [11].

Definition 7 *A* **possibility distribution** Π_A *for A defined on U is represented by a* **fuzzy set F** *on U whose membership function μ_F is identical to the possibility distribution function π_A, i.e., ($\mu_F(u) = \pi_A(u)$) ($\forall u \in U$).*

Thus, a possibility distribution over a set U can be used to define a fuzzy set of U, or vice versa [7]. This explains why these two concepts are used interchangeably throughout this paper. Furthermore, possibility distributions subsume the conventional and set-value representation [10].

The notion of α-sets [11] allows us to get from fuzzy to crisp sets. This is useful, for example, if we want to make a binary decision based on imprecise information.

Definition 8 *Given a fuzzy set F over U. Then the α-level set of F, denoted by F_α, is defined by $F_\alpha = \{u \in U \mid \mu_F(u) \geq \alpha\}$.*

A fuzzy set F may be decomposed into its level sets through the resolution identity

$$F = \sum_\alpha \alpha F_\alpha \quad (4)$$

where αF_α is the product of α with the set F_α and $\sum_\alpha$ is the union with α ranging from 0 to 1. The product αF_α is calculated as one might expect, hence, $\alpha F_\alpha = \{\alpha/u_1, ..., \alpha/u_n\}$ with $u_i \in F_\alpha$ for all i. In this paper, we assume that the fuzzy sets are based on non-continuous possibility distributions, and hence the summation in equation (4) is finite.

Finally, the extension principle introduced by Zadeh [11] allows arithmetic operations based on numeric values to be extended to apply to possibility distributions.

Definition 9 *Given a binary operation $\circ$ defined on the elements of a universe of discourse U. Then, the operation $\circ$ can be extended to apply to any two possibility distributions Π_x and Π_y over U by the following:*

$$\begin{aligned} &\Pi_x \circ \Pi_y \\ &= \{\pi_x(u1)/u1 \mid u1 \in U\} \circ \{\pi_y(u2)/u2 \mid u2 \in U\} \\ &= \{min(\pi_x(u1), \pi_y(u2))/(u1 \circ u2) \mid u1, u2 \in U\}. \end{aligned}$$

This extension of arithmetic operations is well-defined, since, by assumption, the operation $(u1 \circ u2)$ is well-defined for u1, u2 $\in$ U and the minimum of two real numbers taken from [0,1] is also well-defined, resulting in a real number again from the [0,1] interval.

4 Extending the Relational Data Model

4.1 The Possibilistic Relational Data Model

Various attempts to enhance the relational database model by fuzzy extensions can be found in the literature [1, 7, 12, 10, 8, 5]. This section describes one of them,

called the possibilistic relational data model (PRDM) [10], which serves as the basis for the work presented in this paper. A relation in PRDM is defined as follows:

Definition 10 *Let A_i for i from 1 to n be attributes defined on the domain sets U_i, respectively. Then a* **possibilistic relation** *r is defined on $R(A_1, A_2, ..., A_n)$ as a subset of the Cartesian product of a collection of possibility distributions:*

$$r \subseteq P(U_1) \times P(U_2) \times ... \times P(U_n)$$

where $P(U_i)$ denotes the collection of all possibility distributions on U_i.

Each tuple t of r consists of a Cartesian product of possibility distributions on the respective U_i's, i.e., t[A_i] = $\Pi(A_i)$ where $\Pi(A_i)$ is a possibility distribution on U_i.

PRDM allows for data values that can be modeled by possibility distributions. This includes multiple values (e.g. $\{23, 24, 25\}$), discrete possibility distributions (e.g. $\{0.7/130, 0.8/135\}$), linguistic terms as labels for fuzzy sets (e.g. *young*, *about* -20, *light*), and single values (e.g. 140). For example, the fuzzy set *light* could be represented as $\{1.0/100, 0.9/110, 0.7/120\}$ as depicted in the possibilistic relation in figure 3.

Name	Age	Weight	Sex
Uwe	$\{23, 24, 25\}$	130	male
Anita	*about* $-$ 20	*light*	female
Hans	*young*	$\{0.6/120, 1.0/130\}$	male
Mary	20	110	female

Figure 3: The person relation: a possibilistic relation

4.2 Relational Algebra for the PRDM

Several suggestions can be found in the literature on how to extend the relational algebra operations to deal with possibilistic data [7, 12, 10]. We limit our discussion here to the SELECT operation. (Other relational algebra operations may be found in [8, 10].) The SELECT operation allows for the specification of an approximate match.

Definition 11 *The syntax of the SELECT operation is*

$$SELECT(r\ WHERE\ r.A_i\ is\ F) \qquad (5)$$

where F refers to a fuzzy set defined over the domain U_i of the attribute A_i. The query is evaluated by measuring the agreement of each tuple in the relation r with F. This possibility measure [7] is defined by

$$Poss(t[A_i]\ is\ F) = \max_{u \in A_i} \min(\pi_{A_i}(u), \mu_F(u)) \quad (\forall u \in U_i) \qquad (6)$$

The result of a selection operation is a set of tuples, each with an associated measure of how it satisfies the query. It is, in general, useful to specify a threshold of acceptance $\alpha \in [0,1]$ to select all tuples that match the selection criteria to at least that degree α (see the α-level set concept of definition 8).

Lemma 1 *If the data is crisp, e.g., $t[A_i] = u = \{1.0/u\}$, then equation 6 simplifies to the following possibility measure for a tuple t of r:*

$$Poss(t[A_i] is F) = min(1.0, \mu_F(u)) = \mu_F(u). \qquad (7)$$

For possibilistic data but a crisp selection equation 6 defaults to

$$Poss(t[A_i] = u) = min(\pi_{A_i}(u), 1.0) = \pi_{A_i}(u). \qquad (8)$$

A selection condition comparing two imprecisely specified attribute values is evaluated similarly.

Definition 12 *Given the attributes A_i and A_j of relation r. To find all tuples with matching attribute values for A_i and A_j we write*

$$SELECT(r\ WHERE\ r.A_i = r.A_j) \qquad (9)$$

This is evaluated by measuring the agreement of the two attribute values for each tuple in the relation:

$$Poss(t[A_i] = t[A_j]) = \qquad (10)$$

$$\max_{u \in dom(A_i) \cup dom(A_j)} \min(\pi_{A_i}(u), \pi_{A_j}(u))).$$

This evaluation could be extended to incorporate the similarity between domain values [10], or, by applying additional fuzzy modifiers, such as, the modifier *very* to the fuzzy set *old* [12].

5 Extending FRQLs with Aggregates

In this section, it is investigated how the aggregate operators presented in section 2 can be redefined to cope with the possibilistic representation of data. The following three cases are to be considered:

- the data over which the aggregate is to be evaluated could be crisp or possibilistic;
- the selection of tuples considered for the aggregation could be precise or vague;
- the data on which the partition is to be based could be crisp or possibilistic.

The third case applied only to aggregate functions, while the others must be considered for both, scalar aggregates and aggregate functions. An important goal of this research in all of the above cases is to satisfy two principles: *consistency:* the generalized operations should default to the crisp operations for conventional data;

and *completeness:* the operations should be well-defined for the fuzzy case [1]. These principles require the generalized versions of aggregates to be natural extensions of their crisp counterparts.

5.1 Aggregate Evaluation of Vague Queries on Crisp Data

In the following, the aggregate operators are extended to cope with approximate queries (fuzzy predicates) on crisp data. An example is the query "What is the average of the *high* salaries of all professors?". More generally, the queries have the form

$$f((A_i)(SELECT(r\ WHERE\ \mathbf{r.A_j\ is\ F}))) \quad (11)$$

where F refers to a fuzzy set defined over the domain of the attribute A_j. It is not necessary that i $\neq$ j as the above example query indicates. F could be a complicated expression involving fuzzy modifiers or a conjunction/disjunction of several fuzzy sets.

According to lemma 1, the evaluation of "$t[A_j]$ *is F*" results in $\mu_F(u)$ if $t[A_j]$ = u. This value indicates the degree to what the tuple matches the selection criteria. It implies that the different tuples should participate to different degrees in the evaluation of the aggregate. The number of tuples to be considered to be in agreement with the selection depends on the chosen level of acceptance, denoted by α. Recall that α is inversely proportional to the number of elements able to satisfy α. We introduce the notion of an α-level relation, based on which we then define how a scalar aggregate is to be evaluated in a fuzzy query.

Definition 13 *Let r be a relation defined over the relation schema $R(A_i, A_j, ...)$ and F be a fuzzy set over the domain of A_j. Let $\alpha \in [0,1]$. Then the α-level relation $A_j^r(\alpha)$ is defined by:*

$$A_j^r(\alpha) = \{t \mid t \in r \wedge \mu_F(t[A_j]) \geq \alpha\} \quad (12)$$

This is based on the concept of an α-level set F_α (definition 8). For a given $\alpha \in$ [0,1] all tuples which satisfy the proposition "r.A_j is F" to at least the degree α are collected in $A_j^r(\alpha)$.

Definition 14 *Given r, A_j, and F from definition 13. A query of the form shown in equation 11 is evaluated by calculating the aggregate f on $A_j^r(\alpha)$ for all α and associating α with the result. The semantics are:*

$$f((A_i)(SELECT(r\ WHERE\ r.A_j\ is\ F))) =$$

$$\{\mu_f(y)/y \mid \mu_f(y) = sup_\alpha\{f((A_i)(A_j^r(\alpha))) = y\}\}. \quad (13)$$

The fuzziness of F induces a fuzzy set of possible answers instead of one value. The following example is given to illustrate the above definition.

Position	$\mu_{Prestigious}$
Assistant	*0.5*
Associate	*0.8*
Full	*1.0*

Figure 4: The fuzzy set Prestigious

Example 2 *Let Prof be the relation defined in figure 1. Let Prestigious be a fuzzy set defined on the set of different positions shown in figure 4.*

The query "What is the average salary of employees holding prestigious positions?" is formally expressed by avg((Salary)(SELECT(Prof WHERE Prof.Position IS Prestigious))). This query is evaluated by constructing α-level relations (see definition 13 and 14). Let t1 be the first tuple in the Prof relation, t2 the second, etc.

For α = 1.0, $Position^{Prof}(\alpha) = \{t2, t3\}$ by equation 12. Then, avg((Salary) ($Position^{Prof}$ (1.0))) = avg(4500, 4000) = 4250. Similarly, for α = 0.8, avg((Salary) ($Position^{Prof}$ (0.8))) = 3625, and for α = 0.5, avg((Salary) ($Position^{Prof}$ (0.5))) = 3600. Thus the result of the query is the fuzzy set $\{1.0/4250, 0.8/3625, 0.5/3600\}$.

The threshold level α determines which values are taken into consideration for the evaluation. The smaller α is the more elements are included in the evaluation.

Lemma 2 *Let r be a relation defined on a relation schema containing the attributes A_i and A_j. If $\alpha_1 \leq \alpha_2$ then $A_j^r(\alpha_2) \subseteq A_j^r(\alpha_1)$. This implies that $max(((A_i)A_j^r(\alpha_1)) \geq max(((A_i)A_j^r(\alpha_2))$. Correspondingly, $min(((A_i)A_j^r(\alpha_1)) \leq min(((A_i)A_j^r(\alpha_2))$ and also $sum(((A_i)A_j^r(\alpha_1)) \geq sum(((A_i)A_j^r(\alpha_2))$.*

There is no monotonicity for the other aggregates [6]. It may be of interest to summarize the result of such a query in a more concise way. A simple approach is to give an α value which one considers to be a sufficient matching degree; then the result of applying the aggregate to $A^r(\alpha)$ would be returned. While the discussion in this section has concentrated only on scalar aggregates, the approach proposed here can be directly extended to aggregate functions. This is done by first partitioning the relation, which produces precise partitions, since the underlying data is crisp. Then, each partition can be treated as outlined above.

5.2 Evaluation of Scalar Aggregates on Possibilistic Data

The queries considered in this section correspond to the ones described in definition 3, however, possibility distributions are now allowed as attribute values instead of just simple constants. The proofs of consistency are omitted, but can be found in [9].

5.2.1 Generalized Count Aggregate

The count aggregate, defined in definition 3, returns the number of tuples of a relation. One can directly adopt this definition for the possibilistic case. This is referred to as *fcount1*. An alternative is the use of the sigma-count operation [11], which is defined as follows.

Definition 15 *Given a fuzzy subset F over U, i.e.,* $F = \{\mu(u_1)/u_1, \mu(u_2)/u_2, ..., \mu(u_n)/u_n\}$. *Then, the cardinality of F, called sigma-count, is the sum of the grades of memberships in F. Thus, sigma-count(F)* $= \sum_{\forall i} \mu(u_i)$.

We propose to use the sigma-count as an extended count aggregate, here called *fcount2*.

Definition 16 *Given a relation r defined on the relation schema R(..,A,...). Let A be defined on the domain* $U = \{u_1, ..., u_n\}$. *The relation r consists of tuples* t_i *with* $1 \leq i \leq m$. *Let* $t_i[A] = \{\mu_i(u_1)/u_1, ..., \mu_i(u_n)/u_n\}$ $(\forall i)$. *Then the fcount2 aggregate on the attribute A of r is defined by* $fcount2((A)(r)) = \sum_{i=1}^{m} \sum_{j=1}^{n} \mu_i(u_j)$.

The result of a *fcount2* operation is a real number, but it is understood that the result may be rounded, if necessary. The *fcount2* operation does not exhibit all features of the conventional count operation, i.e., it does not always return the same value for the different attributes of a relation. If this characteristic is required, then the designer will choose the *fcount1* operator over the *fcount2* aggregate. The *fcount2* operator has been proven useful for defining the generalized average operator (see section 5.2.5).

Example 3 *The fcount1 for the Weight attribute of the person relation in figure 3 is 4, since there are four tuples. Whereas, fcount2((Weight)(person)) = 6.2.*

5.2.2 Generalized Sum Aggregate

The sum aggregate is is only defined for numeric domains. Necessary characteristics of the extended sum aggregate are commutativity and associativity, since the relational data model does not impose any order on the tuples of a relation [2]. The sum aggregate, here termed *fsum*, is based on the extension principle (definition 9).

Definition 17 *Given r, R, A, U and the tuples* t_i *from definition 16. The tuples* t_i *are assumed to take on possibilistic values for the attribute A, i.e., for i from 1 to m we have* $t_i[A] = \{\mu_i(u_{ki})/u_{ki} \mid 1 \leq ki \leq n\}$. *The fsum aggregate of the attribute A of a relation r is defined by* $fsum((A)(r)) = \{u/y \mid ((y = \sum_{ki=k1}^{km} u_{ki}) \wedge (u = \min_{i=1}^{m} \mu_i(u_{ki}))) (\forall k1, ..., km : 1 \leq k1, ..., km \leq n)\}$.

It is fairly straigthforward to see that the *fsum* operation is commutative as well as associative, since both the summation and the minimum operation are. The result of the *fsum* aggregate is in general a possibility distribution. An example of the fsum aggregate operator is given next.

Example 4 *The fsum aggregation of the Weight of the relation person of figure 3 is fsum((Weight)(person))* $= \{0.6/460, 1.0/470, 0.9/480, 0.7/490\}$. *This result states that the sum of all values is 470 with the possibility of 1.0, and that values close to 470 are also possible results.*

5.2.3 Generalized Max Aggregate

The max aggregate is defined for any domain that has an order defined on it. The extension of the max aggregate to deal with possibility distributions, here called *fmax*, is again based on definition 9. Consequently, commutativity and associativity of the operation are guaranteed.

Definition 18 *Given r, A, U and* t_i*'s from the previous definition. The fmax scalar aggregate is defined by* $fmax((A)(r)) = \{u/y \mid ((y = \max_{ki=k1}^{km} u_{ki}) \wedge (u = \min_{i=1}^{m} \mu_i(u_{ki}))) (\forall k1, ..., km : 1 \leq k1, ..., km \leq n)\}$

This generalization of the maximum aggregate is well-defined for the same reasons that the extension principle is. The result of the *fmax* aggregate is in general a possibility distribution. An example of the *fmax* operation is given below.

Example 5 *The fmax of the Weight attribute of relation person of figure 3 is fmax((Weight)(person))* $= \{0.6/130, 0.6/130, 0.6/130, 1.0/130, 0.9/130, 0.7/130\} =$ 130 *This is a realistic result, since 130 is indeed the maximum value.*

5.2.4 Generalized Min Aggregate

The min aggregate is extended for the possibilistic data models in the same manner as the max aggregate, except for replacing the symbol max in definition 18 by min.

Example 6 *The fmin of the Weight attribute of relation person of figure 3 is fmin((Weight)(person))* $= \{1.0/100, 0.9/110\}$. *This is intuitive, since 100 and 110 are among the lowest values of the Weight attribute.*

5.2.5 Generalized Avg Aggregate

One could define the generalized average aggregate in terms of the quotient of the generalized sum and count aggregates. We take an alternative route based on the possibilistic expected value, PEV, a concept introduced by Zemankova and Kandel [12].

Definition 19 *Let A be an attribute of a numeric domain U. Let $\pi(u_i)$ be the possibility distribution for value u_i, and n the number of $\pi(u_i)/u_i$ pairs for the attribute A. The PEV for attribute A is defined by*

$$PEV(A) = \frac{\sum_{i=1}^{n} \pi(u_i) * u_i}{n}$$

We propose to define the average aggregate *favg* in terms of the PEV operation by replacing the denominator n by *fcount2*.

Definition 20 *Given r, A, U and the tuples t_i from definition 16. Now, the favg operator is defined to be*

$$favg((A)(r)) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} \mu_i(u_j) * u_j}{fcount2((A)(r))}.$$

The *favg* operator results in a real number, and may be rounded, if necessary. The *favg* operator is demonstrated by an example below.

Example 7 *Given the relation in figure 3. Since fcount2((Weight)(person)) = 6.2, we have favg((Weight)(person))* $= (1*100+0.9*110+0.7*120+1*130+0.6*120+1*130+1*110)/6.2 = 725/6.2 = 116.9$

Finally, the conventional *any* operator can be directly adopted from definition 3 since it tests whether there is a tuple in the relation or not, and thus does not consider the actual content of the relation.

5.3 Aggregate Evaluation of Aggregate Functions on Possibilistic Data

As was outlined in section 2.3, aggregate functions are based on the evaluation of scalar aggregates. This also holds for aggregate functions in the PRDM. If the attribute values on which the partition is based are precise values then the extensions to be made for the possibilistic relational database are straightforward as shown in the next section. The case where attribute values on which the partition is based are possibilistic is presented in section 5.3.2.

5.3.1 Partitioning on Precise Data

If the attribute values on which the partition is based consist exclusively of crisp values, then it is in fact possible to directly translate the procedure described in definition 5 by translating the individual operations of step 1, such as selection by WHERE clause, to their corresponding fuzzy counterparts. Thus, it may affect step 1 of the evaluation process described in definition 5. This is because the partition of tuples by the BY clause produces an exact partition. The only other change concerns the scalar aggregates applied in step 3 of the procedure which now are replaced by the generalized scalar aggregates as defined in section 5.2. The following example demonstrates the procedure of evaluating an aggregate function over possibilistic data by partitioning it on precise data.

Example 8 *Let person be the relation given in figure 3. The values for the Weight attribute are possibilistic, but the values for the Sex attribute are all precisely known. The query "What is the average Weight of the persons for each sex?" is formally expressed by favg((Weight)(person) BY Sex). This can be evaluated as follows:*

1. *Step one is not needed.*
2. *First, find the attribute values to partition on by*

   $$PROJECT_{[Sex]}(person) = \{male, female\}.$$

   Thus, by equation (3) there are the following two partitions:

   - $P^{Sex}_{person}(male) = \{(Uwe, ..., 130, male), (Hans, ..., \{0.6/120, 1.0/130\}, male)\}$;
   - $P^{Sex}_{person}(female) = \{(Anita, ..., \{1.0/100, 0.9/110, 0.7/120\}, female), (Mary, ..., 110, female)\}$.
3. *Next, apply the scalar aggregate operator average to each partition:*
   - $favg((Weight)(P^{Sex}_{person}(male))) = favg(\ 130, \{0.6/120, 1.0/130\}) = 127.7$
   - $favg((Weight)(P^{Sex}_{person}(female))) = favg(\ 110, \{1.0/100, 0.9/110, 0.7/120\}) = 109.1$
4. *Associate the result obtained for each partition with the value on which that partition was based. The result relation is depicted in figure 5.*

Sex	*Avg-Weight*
male	*127.7*
female	*109.1*

Figure 5: The avg-weight-by-sex relation

5.3.2 Partitioning on Possibilistic Data

Below, we base the partitioning process on an attribute containing possibilistic data. Under these circumstances, one cannot find a real partition any more. The best one can hope for is to determine to what degree a tuple participates in a given partition. Furthermore, it is possible that tuples participate in and thus contribute to more than one partition. Note the similarity between this situation and the case discussed in section 5.1. For simplicity, we assume that the attribute over which the aggregate is to be evaluated is of crisp nature. The problem of partitioning on possibilistic data

leads to the introduction of a new concept, the α-level partition, which is defined in definition 21. Definition 22, a variation of definition 14, describes how the aggregate function $f((A_i)(r)BY\,A_j)$ is to be evaluated when allowing possibilistic values for A_j while A_i consists of crisp values.

Definition 21 *Let r be a relation defined over the relation schema $R(A_i, A_j, ...)$. Let the attribute A_i be crisp, and the attribute A_j be possibilistic. Let D be the active domain of A_j, i.e. $D = \{a \mid (t \in r) \wedge \mu_{t[A_j]}(a) > 0\}$ denotes the set of values on which the partition is to be based. The partition function P as defined in equation 3 has to be modified to accommodate for membership values, thus, for all $a \in D$ we have:*

$$P_r^{A_j}(a) = \{t \mid (t \in r) \wedge (a \in D) \wedge (\mu_{t[A_j]}(a) > 0)\} \quad (14)$$

Let $\alpha \in [0, 1]$. For each partition $P_r^{A_j}(a)$ define the α-level partition $L_r^{A_j}(\alpha, a)$, a function of two variables α and a, by:

$$L_r^{A_j}(\alpha, a) = \{\, t \mid (t \in P_r^{A_j}(a)) \wedge (\mu_{t[A_j]}(a) \geq \alpha)\} \quad (15)$$

Definition 22 *Given r, A_i, A_j and D from the previous definition. Then the query $f((A_i)(r)BY\,A_j)$ is evaluated by computing for each partition $P_r^{A_j}(a)$ the supremum of the aggregate evaluation of all associated α-level partitions $L_r^{A_j}(\alpha, a)$ for all α. More precisely,*

$$f((A_i)(r)BY\,A_j) = \{a \circ F \mid (a \in D) \wedge$$

$$F = \{\mu_f(y)/y \mid \mu_f(y) = sup_\alpha\{f((A_i)(L_r^{A_j}(\alpha, a))) = y\}\}. \quad (16)$$

An example is given next to illustrate the previous definition.

Name	Salary	Position
Tom	3500	$\{1.0/Assistant, 0.8/Associate\}$
Jack	4500	$\{1.0/Full\}$
Julie	4000	$\{1.0/Full, 0.5/Associate\}$
Mary	2500	$\{1.0/Associate, 0.7/Assistant\}$
Frank	3500	$\{1.0/Associate\}$

Figure 6: The Prof relation

Example 9 *Given the relation Prof from figure 6. The query "What is the maximum salary of professors per position" formally expressed by fmax((Salary)(Prof) BY Position) has to be evaluated according to definition 22.*

1. *Again, step 1 is not needed for this example.*
2. *First, find the attribute values to partition on; they correspond to the active domain D which is $D = \{Full, Associate, Assistant\}$. This results in three partitions according to equation (14). Each tuple in a partition has associated with it a membership value indicating to what degree it participates in that partition:*
   - $P_{Prof}^{Position}(Full) = \{(Jack, ...)$ *with* $1.0;$ $(Julie, ...)$ *with* $1.0.\}$
   - $P_{Prof}^{Position}(Associate) = \{(Tom, ...)$ *with* $0.8;$ $(Julie, ...)$ *with* $0.5; (Mary, ...)$ *with* $1.0;$ $(Frank, ...)$ *with* $1.0\}$
   - $P_{Prof}^{Position}(Assistant) = \{(Tom, ...)$ *with* $1.0;$ $(Mary, ...)$ *with* $0.7\}$
3. *Now, by equation 15, we can determine for all partitions $P_{Prof}^{Position}(a)$ the different α-level partitions $L_{Prof}^{Position}(\alpha, a)$ for each distinct α. Then, we apply the scalar aggregate operator, max, to those α-level partitions $L_{Prof}^{Position}(\alpha, a)$:*
   - *fmax((salary)($P_{Prof}^{Position}$(Full)) :*
     - *for $\alpha = 1.0$: fmax((Salary)($L_{Prof}^{Position}$(1.0, Full)) = max(4500, 4000) = 4500.*
   - *fmax((salary)($P_{Prof}^{Position}$(Associate)) :*
     - *for $\alpha = 1.0$: fmax((Salary)($L_{Prof}^{Position}$(1.0, Associate)) = 3500.*
     - *for $\alpha = 0.8$: fmax((Salary)($L_{Prof}^{Position}$(0.8, Associate)) = 3500.*
     - *for $\alpha = 0.5$: fmax((Salary)($L_{Prof}^{Position}$(0.5, Associate)) = 4000.*
   - *fmax((salary)($P_{Prof}^{Position}$(Assistant)) :*
     - *for $\alpha = 1.0$: fmax((Salary)($L_{Prof}^{Position}$(1.0, Assistant)) = 3500.*
     - *for $\alpha = 0.7$: fmax((Salary)($L_{Prof}^{Position}$(0.7, Assistant)) = 3500.*
4. *The result of the scalar aggregate evaluation for each α-level partition is associated with the respective level α. Finally, combine all these $\alpha/L_{Prof}^{Position}(\alpha,$ a) pairs for all α-level partitions of a given partition $P_{Prof}^{Position}(a)$ to form a possibility distribution. The resulting relation is depicted in figure 7.*

Position	Max-Salary
Full	$\{1.0/4500\}$
Associate	$\{1.0/3500, 0.5/4000\}$
Assistant	$\{1.0/3500\}$

Figure 7: The max-sal-per-position relation

Finally, if both attributes, i.e. the attribute to partition on and the attribute on which to evaluate the aggregate function are possibilistic, then according to definition 22 the result of the third step of the aggre-

gate evaluation process would be of the following form: $0.7/\{1.0/2500, 0.8/3500, 0.5/4000\}$. This can be evaluated by taking the minimum of the membership values, which results in $\{0.7/2500, 0.7/3500, 0.5/4000\}$.

5.4 Vague Queries on Possibilistic Data

We conclude with a discussion of how to combine the aggregate evaluation approach dealing with vague queries (section 5.1) with that for handling possibilistic data (section 5.2 and section 5.3). At this point, we favor the simplification of the evaluation strategy as indicated at the end of section 5.1. More specifically, the user chooses an α value s/he considers to be an acceptable threshold value for an approximate selection. When the user specifies a query involving a vague selection clause, such as "Find all *old* people", the result will be the list of tuples which satisfy that selection criteria with at least the degree α. Hence, the result of a vague selection would be an α-level relation described in equation 12.

Introducing possibilistic data concerns us in as much as the data underlying the selection clause is possibilistic. Then the evaluation of a vague selection clause, such as "Age is *old*", has to be altered. Definition 14 is modified by evaluating the vague selection in accordance with equation 6 instead of lemma 1. This again produces a possibility measure for each tuple describing the degree of matching between the tuple and the selection clause. All tuples with a possibility measure above the threshold α are collected in the respective α-level relation, i.e., they will be considered further in the query evaluation process. In terms of the aggregate evaluation process outlined in definition 5, this means that the vague query part is dealt with in step 1 of the process. A non-base relation (α-level relation) is generated in step 1, based on which the aggregate evaluation process can continue with the remaining steps as discussed in section 5.2 and 5.3.

6 Conclusions

The PRDM [10] is a generalized version of the relational database model capable of capturing precise, as well as imprecise, data. By extending the query languages for the conventional relational database model, the model is able to handle vague queries on both types of data. However, to make use of the PRDM in real world applications, it became imperative to develop and incorporate suitable aggregate evaluation mechanisms, found in most existing relational database systems, such as, System R or Ingres [3].

The major contribution of this paper is the development of a framework of evaluation procedures for scalar aggregates as well as aggregate functions for the PRDM. In particular, our approach handles the application of aggregate operators within vague queries as well as the application of aggregates to possibilistic data found in the PRDM. We show that the handling of aggregate functions in the PRDM can be handled in a natural and intuitive manner. The usefulness of all new operators is demonstrated in a pragmatic manner by intuitive examples.

Possible extensions to this research are manifold. For instance, aggregate operations for fuzzy data models which are not based on the notion of discrete possibility distributions are needed, e.g., interval representations, continuous functional descriptions, or continuous possibility distributions. Also, the question of update operations in possibilistic databases is mostly unexplored. For example, it is not obvious how to perform an operation such as "Update the salaries of all *old* people".

References

[1] Buckles, B. P. and Petry, F. E. A Fuzzy Representation of Data for Relational Databases. *Fuzzy Sets and Systems.* 7, (1982), 213 - 226.

[2] Codd, E. F. A relational model of data for large shared data banks. *Communications of the ACM* 13, 6 (June 1970), 337 - 387.

[3] Date, C. J. *A Guide to Ingres.* Addison-Wesley Pub. Co., Reading, Mass, 1987.

[4] Klug, A. Equivalence of Relational Algebra and Relational Calculus Query languages Having Aggregate Functions. *Journal of ACM* 29, 3 (July 1982), 699 - 717.

[5] Oezsoyoglu, G., Oezsoyoglu, Z. M. and Matos, V. Extending Relational Algebra and Relational Calculus with Set-Valued Attributes and Aggregate Functions. *ACM Trans. on Database Systems* 12, 4 (Dec. 1987), 566 - 592.

[6] Prade, H. Global Evaluations of Fuzzy Sets of Items in Fuzzy Data Bases. *Int. Workshop on Fuzzy System Applications.* Fukuoka, Japan, (Aug 1988).

[7] Prade, H. and Testemale, C. Generalizing database relational algebra for the treatment of incomplete/uncertain information and vague queries. *Information Science* 34, (1984), 115 - 143.

[8] Raju, K. V. S. V. N. and Majumdar, A. K. Fuzzy Functional Dependencies and Lossless Join Decomposition of Fuzzy Relational Database Systems. *ACM Trans. on Database Systems* 13, 2 (June 1988), 129 - 166.

[9] Rundensteiner, E. A. and Bic, L. Evaluating Aggregate Functions on Possibilistic Data. University of California, Irvine, Information and Computer Science Department, Tech. Rep. 89-12, May 89.

[10] Rundensteiner, E. A. The Development of a Fuzzy Temporal Relational Database (FTRDB): An Artificial Intelligence Application. Master's thesis. Dept. of Computer Science. Florida State University. 1987.

[11] Zadeh, L. Fuzzy Sets. *Information and Control* 8, (1965), 338 - 353.

[12] Zemankova, M. and Kandel, A. *Fuzzy Relational Database - A Key to Expert Systems.* Koeln: Verlag TNV Rheinland, 1984.