Evaluating Aggregate Functions on Possibilistic Data

Elke A. Rundensteiner and Lubomir Bic

Department of Information and Computer Science
University of California, Irvine
May, 1989

Technical Report 89-12

Evaluating Aggregate Functions on Possibilistic Data

ELKE A. RUNDENSTEINER and LUBOMIR BIC

Department of Information and Computer Science

University of California, Irvine

May, 1989

Abstract

The need for extending information management systems to handle the imprecision of information found in the real world has been recognized. Fuzzy set theory together with possibility theory represent a uniform framework for extending the relational database model with these features. However, none of the existing proposals for handling imprecision in the literature has dealt with queries involving a functional evaluation of a set of items, traditionally referred to as aggregation. Two kinds of aggregate operators, namely, scalar aggregates and aggregate functions, exist. Both are important for most real-world applications, and are thus being supported by traditional languages like SQL or QUEL. This paper presents a framework for handling these two types of aggregates in the context of imprecise information. We consider three cases, specifically, aggregates within vague 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, the sigma-count operation, and the possibilistic expected value. The consistency and completeness of the proposed operations is shown.

Categories and Subject Descriptors: H.2.3 [**Database Management**]: languages - *query languages.*

Additional Key Words and Phrases: Scalar Aggregates, Aggregate Functions, Partitioning Function, Relational Database Model, Possibilistic Relational Model, Possibility Theory and Fuzzy Set Theory.

1 INTRODUCTION

It has been widely recognized that the vagueness, uncertainty, and incompleteness inherent in the real world data has to be dealt with in information management systems. Research in coping with this phenomenon has to a large extend been based on the relational data model developed by Codd [6]. Unfortunately, all the available implementations of relational database systems are modeling the real world in a deterministic manner and allow only for exact retrievals.

There have been several attempts in the literature to use fuzzy set theory as proposed by Zadeh [23] and related concepts for providing a suitable interpretation of different types of impreciseness and vagueness in relational database models. The emphasis is on the explicit representation of fuzziness in a system rather than trying to eliminate or disguise it by some clever trick or to simply ignore it and oversimplify the modeling process unrealistically.

The two major objectives of these efforts are enhancements to the data model, i.e. the problem of representing incomplete and uncertain data, and the development of new retrieval techniques, i.e. the question of how to access this data. The first issue mainly addresses the limitation of the conventional data model to allow attributes to take but one constant value from a base set (domain). This restriction has been modified by different approaches presented in the literature. Buckles and Petry [3] have suggested to replace attribute values by sets of values. The concept of a similarity measure was introduced to identify when tuples were similar enough to be redundant. Umano [22] and Zvieli modified this restriction by allowing fuzzy sets and memberships values. Finally, Umano [21], Prade and Testemale [14] and Dubois and Prade [9] have proposed models based explicitly on possibility distributions where domain values as well as associations among entities are represented by possibility distributions.

The second issue is that the enhanced data models require the development of new relational query languages (RQLs) capable of coping with the different types of fuzzy rep-

resentations of data. Most of these attempts to design an enhanced RQL, here called a FRQL, are based on a form of the generally accepted relational algebra or the relational calculus, both developed by Codd [6]. Prade and Testemale [14] extended the classical relational algebra to accommodate the possibilistic representation. They also showed that different types of 'null values' used in classical relational database systems can be obtained for free in their model. Zemankova and Kandel [26] presented a thorough discussion of fuzzy relational database models. Rundensteiner, Bandler et al [18] advocate the use of resemblance relations instead of similarity relations to measure the nearness between tuples and to determine whether tuples are redundant.

Queries solely composed of retrieval operations, such as the relational algebra operations [6], are inadequate for many important applications of database management systems [11]. Practically all real-world problems need query capabilities involving the application of aggregate and statistical functions to database relations. The query "What is the *total* number of students?" is an example of a query which cannot be expressed with relational algebra. Consequently, most commercially available systems provide a set of these aggregate operators [20, 7]. This strongly suggests that the evaluation of aggregates has to be dealt with also in the context of the extended relational models. However, most research on extending the relational query languages has concentrated on the relational algebra operations, and has neglected the issue of aggregate evaluation.

This paper investigates the incorporation of aggregate operators into Fuzzy Relational Query languages. More precisely, the paper addresses the problem of evaluating queries asking for some functional evaluation of a set of items within an extended relational database. Examples of such queries are "What is the *maximum* salary of professors at UCI?" and "What is the *average* salary of the employees for each sex?". The first query is an example of a so called scalar aggregate whereas the second one is referred to as aggregate function. Both kinds of queries can be managed by traditional query languages like SQL [1] or QUEL

[7]. When the relational model is extended to model the imprecision and vagueness of information found in the real world, then the definition of these aggregate operations must also be extended. There are several general cases we consider. First, we handle approximate queries on a relational database containing precise information. An example of this 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. In this case it is important to keep track of the influence of the fuzziness on the possible values of the answer. Thus, the result of such an operation is not necessarily one single value, but a set of results annotated by their respective possibility. We introduce the notion of an α-level relation, a concept which is closely related to the concept of α-level sets found in fuzzy set theory [23].

Alternatively, the information stored in the database may be of possibilistic nature, and thus the aggregate evaluation has to deal with this imprecision in the data. An example of this second case is the query "What is the maximum salary of professors at UCI?" with the salary being imprecisely specified in the database, e.g. it is equally possible that the salary of John is 3,000 or 3,500. This case can further be subdivided into several subproblems. First, we develop a framework for the evaluation of scalar aggregates on possibilistic data, an example of which is the just presented query. Our approach makes use of various concepts developed within the fields of fuzzy set and possibilistic theory, such as, the extension principle introduced by Zadeh [23], the sigma-count operation [24], and the concept of a possibilistic expected value, which had been developed by Zemankova and Kandel [26] to cope with null values in relational databases. This framework constitutes the foundation for the remainder of the research.

Next, the problem of evaluating functional aggregates on possibilistic data is addressed. The strategy of decomposing the aggregate evaluation process for functional aggregates into several simple steps forms the basis of our work on aggregate functions. We approach this problem by first considering the case of partitioning on precise data while evaluating the

aggregate on imprecisely known data. This case essentially reduces to the application of the extended scalar aggregates to precise partitions. In order to partition on possibilistic data, however, we introduce the notion of an α-level partition. An α-level partition enhances the notion of partitioning functions with the concept of α-level sets found in fuzzy set theory.

The two approaches are then combined to handle a database containing possibilistic specified data and queries with approximate restrictions. Let us mention here that the solutions we propose in this paper satisfy two principles, namely, the consistency and the completeness requirement.

The paper is structured in the following manner. It starts with a review of the classical relational data model and relational query languages. Then, aggregate operators as used in conventional database models are discussed, with particular emphasis on the distinction between scalar aggregates and aggregate functions (section 3). A stepwise evaluation process for aggregate functions is being proposed. After having presented the basics of fuzzy set theory in section 4, a short overview of the possibilistic extensions of the relational database model as well as the relational language defined on the extended relational model is given in section 5. Special emphasis is put on the possibilistic relational model since the research discussed in the paper is based on this model. A more thorough presentation of the model can be found in [Re87]. Finally, section 6 is devoted to our proposal for evaluating aggregates on the possibilistic relational data model. The different cases as previously described are analyzed in detail. In section 6.1, we discuss our approach of evaluating aggregates within vague queries. The evaluation of scalar aggregates for possibilistic data is presented in section 6.2. Throughout this section we point out the relationship of our approach to concepts such as the extension principle [23], the sigma-count operation [24], and the concept of a possibilistic expected value [26]. Finally, based on the results presented in section 6.2, the steps of the aggregate evaluation process for functional aggregates are extended in a coherent manner to accommodate for possibilistic data. We approach this problem by first considering the case of partitioning on precise data while evaluating the aggregate

function on imprecisely known data. This straightforward extension is described in section 6.3.1. In order to partition on possibilistic data, however, we introduce the notion of an α-level partition. Section 6.3.2. then handles the case of partitioning on possibilistic data. Finally, to base the partitioning as well as the actual aggregate evaluation on possibilistic data falls into place. This work is based on our initial work presented in [16].

2 THE CLASSICAL RELATIONAL MODEL

In this section some basic concepts related to the classical relational database model [6, 5] are introduced.

Attributes A_i are symbols from a finite set A. Each attribute A_i has associated with it a *domain* denoted by U_i, which is the set of possible values for that attribute.

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 be the union of the sets* U_i, $1 \leq i \leq n$, *which are the domains of the attributes* A_i, *respectively. Then a* **relation** *r on the relation schema R is defined as a finite set of mappings* $\{t1, t2, ..., tp\}$ *from R to U with the restriction that for each mapping* $t \in r$, $t[A_i]$ *must be in* U_i, $1 \leq i \leq n$, *where* $t[A_i]$ *denotes the value of tuple t on attribute* A_i. *These mappings are called* **tuples**. *The size of R is also called the degree of the relation r, or short, deg(r).*

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.

The formalism of mappings is used to avoid any explicit ordering of the attribute names in the relation schema. Rather, a tuple is a set of values, one for each attribute name in the relation schema and the associated relations r_i. If X consists of some domains U_i of the relation schema R, then the notation t[X] denotes the restriction of the tuple t to the attributes captured by X. For example, given the tuple abc in R(A,B,C), then t[A,B] = ab.

A simple view of a relation is the table format, where each row represents a tuple and each column corresponds to one attribute. All items in a column consist of values from the same domain. Consequently each tuple within a relation has the same set of attributes. All rows, called tuples, are distinct; duplicates are not allowed. Each relation consists of a relation name, a nonempty set of attributes with corresponding domains (the relation schema), a key [6], and a (possibly empty) set of tuples.

Definition 2 *A* **key** *is a nonempty set A of attribute names of the relation schema R which identifies the tuples uniquely, i.e., at any time, no two tuples in r can have the same values on all attributes in A; and no attribute name can be discarded from the key without destroying the uniqueness condition.*

In general, there are two types of languages defined for the relational database model. These languages are the relational algebra and the relational calculus, both proposed by Codd [6]. languages for expressing queries in the relational model are called the data manipulation languages (DMLs). Since these two languages have been shown to be equivalent in expressive power, we will concentrate on one of them, namely, the relational algebra.

There are three relational algebra operations on relations that are of interest to us: projection, Cartesian product, and selection. In the following a notation similar to the one of QUEL [7] is chosen over a more formal mathematical one in order to make the queries more understandable.

Definition 3 *Let r be a (base or derived) relation on the relation schema* $R(A_1, ..., A_n)$. *Let* $X \subset R$ *of size k. Then the* **projection** *of relation r on X, denoted by* $PROJECT_X(r)$, *is a relation on X of degree k. The semantics of this expression are defined by*

$$PROJECT_X(r) = \{t[X] \mid t \in r \wedge t[A_i] = t[X][A_i](\forall A_i \in X)\} \qquad (1)$$

Definition 4 *Given two relations r1 and r2 with deg(r1) = d1 and deg(r2) = d2. Then the* **Cartesian product** *, denoted by (r1 × r2) is a relation of degree (d1 + d2). The semantics are:*

$$(r1 \times r2) = \{t \mid (t = t1 \circ t2) \wedge (t1 \in r1) \wedge (t2 \in r2) \; where \; \circ \; denotes \; concatenation\} \quad (2)$$

Definition 5 *Let r be a (base or derived) relation on the relation schema $R(A_1, ..., A_n)$. Let A_i and A_j be attributes in R, for $1 \leq i, j \leq n$, which are defined on compatible domains. Let c be a value from the domain U_i. Then the* **selection** *on relation r, denoted by SELECT(r WHERE $r.A_i \; \Theta \; r.A_j$), is a relation of degree n defined by:*

$$SELECT(r \; WHERE \; r.A_i \; \Theta \; r.A_j) = \{t \mid t \in r \wedge t[A_i] \; \Theta \; t[A_j]\} \quad (3)$$

with $\Theta \in \{=, <, >, \leq, \geq\}$. The expression SELECT(r WHERE $r.A_i \; \Theta$ c) is defined equivalently.

3 AGGREGATES IN RELATIONAL QUERY languages (RQLs)

In this section, it will be shown how aggregates are handled in conventional relational query languages. Retrieval statements composed out of retrieval operations, such as the relational algebra operations described in previous section, are inadequate for many important applications of database management systems [11]. Many real-world queries involve the application of aggregate and statistical functions to database relations. The query "What is the *total* number of students?" is an example of a query which can not be expressed with relational algebra. Consequently, most commercially available systems provide a set of these aggregate operators [20, 7]. The most common aggregate operators of the conventional RQLs as, for example, found in Ingres [7], are *count, any, sum, avg, min,* and *max.* An informal specification and examples of these aggregates follow:

- *count:* This counts the number of values that exist for a given attribute in a relation. (*e.g., Get the total number of professors currently employed at UCI.*)

- *sum:* This computes the sum of the values present for a given attribute. (*e.g., What is the sum of all salaries UCI spends on professors?*)

- *min:* This returns the smallest of the values present for a given attribute. (*e.g., What is the lowest salary for a full professor?*)

- *max:* This returns the largest of the values present for a given attribute. (*e.g., What is the highest salary for an assistant professor?*)

- *avg:* This calculates the arithmetic mean of the values present for a given attribute, where arithmetic mean is defined to be the sum divided by the count. (*e.g., What is the average salary for female professors?*)

- *any:* This checks whether the relation is empty (0) or not (1). (*e.g., Is there a female professor at UCI?*)

Note, that these operators are arithmetic in nature, and hence can only be computed on numeric arguments.

Both QUEL [7] and SQL [1], the query languages of Ingres and System R, respectively, require that aggregate operations be able to accept arguments with duplicates. This means that they provide two other operators, let us call them UNIQUE and DUP, which are used in conjunction with the just described aggregate operators. These two operators determine whether to perform the aggregation on a multiset of values (e.g. *DUP-sum*) or whether to eliminate duplicates first (e.g. *UN-sum*). For example, to sum the salaries in a professor relation with the *DUP-sum* function one would project the relation on the salary attribute, duplicates would be retained in the result, and the projection would be sent to the sum function. This notion of 'duplicates' not only violates the set-theoretic foundation of the

relational model, but provides various other disadvantages discussed in [11]. Hence, we adopt the simple solution of applying an aggregate on a relation instead of an isolated column, i.e. set.

Note, that aggregate operations may be invoked by themselves, e.g. "What is the total number of students?" or embedded within other clauses of a query, e.g. "How many students are there this term who take more than four courses?". This paper, however, is concerned with the definition of aggregation in a fuzzy environment as such, and thus neglects the nesting of these operations within a query. This decision does not represent any limitations of our framework as shown in the next section.

Another point of interest is that there are, in general, two types of aggregate queries supported by RQLs, namely, the *scalar aggregates* and the *aggregate functions* [20]. The discussion of aggregate functions is postponed until section 3.2, whereas scalar aggregates are described in the following.

3.1 Scalar Aggregates

Scalar aggregates take a set of tuples (a relation) as an argument and produce a single simple value as a result. The following describes the syntax we propose for scalar aggregates as well as the associated semantics.

Definition 6 *Let r be a (base or derived) relation on the relation schema $R(A_1, ..., A_n)$. The general syntax for a scalar aggregate f on the attribute A_i of relation r is $f((A_i)(r))$. The semantics of this are defined by*

$$f((A_i)(r)) = y \text{ with } \quad y = f\{t[A_i] \mid t \in r\} \tag{4}$$

The result of the aggregation is a numeric constant.

By their very nature, aggregates operate on the entire relation, but a selection operation can restrict them to operate locally, i.e. only on certain tuples of the relation. In

other words, the relation r in the expression $f((A_i)(r))$ could be a source - as well as a derived relation. A precise definition of the different aggregate operators which are generally supported in conventional database models is provided next.

Definition 7 *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.*

- $count((r.A)(r)) = n$
- $sum((r.A)(r)) = \begin{cases} \sum_{\forall t \in r} t[A] & n > 0 \\ 0 & n = 0 \end{cases}$
- $min((r.A)(r)) = \begin{cases} \min_{\forall t \in r} t[A] & n > 0 \\ 0 & n = 0 \end{cases}$
- $max((r.A)(r)) = \begin{cases} \max_{\forall t \in r} t[A] & n > 0 \\ 0 & n = 0 \end{cases}$
- $avg((r.A)(r)) = \begin{cases} \frac{1}{n} \sum_{\forall t \in r} t[A] & n > 0 \\ 0 & n = 0 \end{cases}$
- $any((r.A)(r)) = sign(n) \ \ where \ \ sign(n) = \begin{cases} +1 & n > 0 \\ 0 & n = 0 \end{cases}$

For the case n = 0, most implementations define the aggregates *sum, avg, min* and *max* to be zero instead of undefined in order to be able to continue evaluating a query. Nonetheless, it would be more consistent with reality if they return a special null value for those cases. This is an important issue worth further investigating, but it is beyond the scope of this work. It is related to the problem of handling null values in the traditional relational database model [5]. Let us mention here, that conventional systems which allow null values remove those from the column before computing the aggregates, e.g. see Ingres [7].

Attribute variables appearing as an argument for a scalar aggregate are purely local to it and thus do not interfere with any variable outside the scope of the aggregate. The following example demonstrates the point.

Example 1 *Let the relation Prof be defined on the relation schema (Prof.Sal, Prof.Name, ...). Then the query "Find all professors who have a better than average salary" can formally be expressed as:*

$$result = SELECT(Prof\ WHERE\ (Prof.Sal > avg((Prof.Sal)(Prof))))$$

This corresponds, however, to the sequence of two queries listed below:

$$sal\text{-}avg = avg((Prof.Sal)\ (Prof));$$
$$result = SELECT(Prof\ WHERE\ (Prof.Sal > sal\text{-}avg))$$

This demonstrates that, for example, the attribute variable Prof.Sal within the aggregate expression is independent from the one in the remainder of the SELECT expression.

The example demonstrates that a scalar aggregate returns a single scalar value and is independent of the rest of the query. Hence, it can appear wherever a single scalar constant is allowed, e.g. in a select clause, as argument to another aggregate, etc. The aggregate is simply calculated and replaced by its value. This allows us to define and handle scalar aggregates independently of any existing query language.

3.2 Aggregate Functions

As indicated previously, scalar aggregates are aggregations over the entire set of tuples which yield one single value as result. Aggregate functions, on the other hand, compute aggregation over one or more subsets of a relation instead of the relation as a whole. Aggregate functions first partition the tuples of a relation on the values of some attribute of the relation schema, and then compute the aggregation separately for each partition.

Thus, the result of an aggregate function is a relation whose number of tuples equals the number of initial partitions, i.e. the result tuples consist of the attribute value on which the partition has been preformed and its associated aggregate value for each partition. An example for the former type of aggregation is the query "What is the average salary of professors?", whereas for the latter it is "What is the average salary of a professor *for each sex*?". Clearly, the result of the second query consists of two aggregate values based on the bipartition through the sex attribute, i.e. an average salary of all male professors and an average salary of all female professors. The following describes the syntax we propose for aggregate functions as well as the underlying semantics as proposed in [11].

Definition 8 *Let r be a (base or derived) relation on the relation schema* $R(A_1, ..., A_n)$. *An aggregate function is a more generalized form of aggregation than a scalar aggregate, and thus the syntax presented in definition 6 is extended by a BY clause. The general syntax for an aggregate function f on the attribute* A_i *of relation r is defined to be* $f\ ((A_i)(r)\ BY\ A_j)$ *with* $1 \leq i,j \leq n$. *The semantics of this are given by*

$$f((A_i)(r)BYA_j) = \{t[A_j] \circ y \mid (t \in r) \wedge (y = f(\{t'[A_i] \mid (t' \in r) \wedge t'[A_j] = t[A_j]\}))\}. \quad (5)$$

The result of the evaluation of an aggregate function is a set of tuples, and not a constant.

The definition in equation 5 can easily be extended to partition on several instead of one attribute by replacing the attribute A_j by the set of attributes X with $X \subset R$. The following is an example of the application of an aggregate function.

Example 2 *Given the relation Prof in figure 1. Assume, for example, that you are interested in the average salary of a Computer Science professor for each rank instead of the overall average salary of all professors. Then you need to use an aggregate function to express the query "What is the average salary of professors in Computer Science for each rank?". Using the just described notation this query can be expressed as follows:*

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

average-Sal = avg((Prof.Sal) (SELECT (Prof WHERE Prof.Department = 'ComputerScience')) BY Prof.Position)

The result of this, an entire set of tuples, one for each distinct value of the attribute identified in the BY clause is depicted in figure 2.

Position	*Avg-Sal*
Assistant	*3500*
Associate	*2500*
Full	*4250*

Figure 2: The Average-Sal relation

Aggregate functions can appear wherever other relational expressions can appear. The evaluation of an aggregate functions, as expressed in equation 5 of definition 8, can conceptually be performed in several stages. This composition facilitates an understanding of the underlying semantics, and it will furthermore serve as a vehicle for the remainder of the paper. In fact, when extending definition 8 to incorporate the impact of imprecise information, we are able to point out exactly which of the evaluation steps have to be adjusted and which remain untouched. The following enumerates these evaluation steps:

Definition 9 *Let r be a (base or derived) relation on the relation schema $R(A_1, ..., A_n)$. Let A_i and A_j be attributes in 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 of the relation r are partitioned by the distinct values of the attribute listed in the BY clause, i.e. A_j.*
3. *The aggregate operator f is applied to each partition P generated in step 2. In other words, a replacement of the aggregate function f by a set of scalar aggregates of the form $f((A_i)(P))$ takes place. These scalar aggregate operations are evaluated as described in definition 7.*
4. *At the end, 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 9 is straightforward for precise (crisp) values of A_j. It can formally be defined as follows. The partition of the relation r on the attribute A corresponds to a function from the values ai of the domain of A to a set of tuples taken from r defined by

$$P_r^A(ai) = \{t \mid t \in r \wedge t[A] = ai\}. \tag{6}$$

This can easily be extended to a partition on a set of attributes instead of just one attribute A, but since it does not offer any further conceptual insight, we limit our discussion to partitioning on one attribute only. The query evaluation process of definition 9 is demonstrated in the following example.

Example 3 *Assume the Prof relation depicted in figure 1. Then according to the evaluation process described in definition 9, the query given in example 2 would be evaluated as follows:*

1. *First all tuples not fulfilling the selection criteria are removed, i.e. the tuple with the name attribute Frank is removed since professor Frank does not belong to the Computer Science department but to the Engineering department.*

2. *Now the attribute values to partition on can be found by*

$$PROJECT_{[Position]}(Prof) = \{Assistant, Associate, Full\}.$$

Thus, by equation (6) there are three partitions of the relation which are:

- $P_{Prof}^{Position}(Assistant) = \{(Frank, 3500, Assistant, ComputerScience)\}$;
- $p_{Prof}^{Position}(Associate) = \{(Mary, 2500, Associate, ComputerScience)\}$;
- $P_{Prof}^{Position}(Full) = \{(Jack, 4500, Full, ComputerScience),$
  $(Julie, 4000, Full, ComputerScience)\}$.

3. *Then, apply the scalar aggregate operator, average, to each partition as discussed in definition 7. For example, avg((Sal)($P_{Prof}^{Position}(Assistant)$)) = 3500.*

4. *The result of this operation for each partition $P_{\tau}^{A}(ai)$ is associated with the attribute value ai on which the partition has been based. For example, the tuple (Assistant, 3500) is formed. This finally results in the relation Average-Sal depicted in figure 2.*

4 BASIC CONCEPTS OF FUZZY SET AND POSSIBILITY THEORY

This section introduces the basic concepts of fuzzy set and possibility theory as proposed by Zadeh [23] and others [9] .

Definition 10 *Let U be a universe of discourse. F is a* **fuzzy subset** *of U, if there is a membership function*

$$\mu_F \mid U \rightarrow [0, 1], \tag{7}$$

which associates with each element $u \in U$ a grade of membership $\mu_F(u)$ in the fuzzy set F. Note, that $\mu_F(u)$ is a real number taken from the interval [0,1].

Zadeh [23] proposed the following notation for a fuzzy set F:

$$F = \{\mu_F(u_1)/u_1, \mu_F(u_2)/u_2, ..., \mu_F(u_n)/u_n\} \tag{8}$$

where $u_i \in U$ for $1 \leq i \leq n$.

Note that a classical subset A of U is a special case of a fuzzy subset with all membership values $\mu_A \in \{0, 1\}$, i.e.,

$$\mu_A(u) = \begin{cases} 1 & if\ u \in A \\ 0 & if\ u \notin A \end{cases} \tag{9}$$

An example of a fuzzy set is given next.

Example 4 *The fuzzy set Old could be defined on the domain Age* $= \{40, 50, 60, 70, 80, 90\}$ *in the manner described in figure 3. The tuple (90,1.0) denotes that the membership of Age=70 in the fuzzy set Old is 1.0, meaning, that anybody with the Age of 70 is considered old.*

Age	μ_{Old}
40	*0.1*
50	*0.4*
60	*0.7*
70	*1.0*
80	*1.0*
90	*1.0*

Figure 3: The fuzzy set Old

A close connection between fuzzy sets and possibility theory has been established [25]. The grade of membership $\mu_F(u)$ of u in the fuzzy set F may be interpreted as the degree of possibility of u given F [14]. This is stated more precisely in the following.

Definition 11 *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.,*

$$\mu_F(u) = \pi_A(u) \text{ for all } u \text{ in universe } U$$

Thus, a possibility distribution over a set U can be used to define a corresponding fuzzy set of U, or vice versa. That is, given a fuzzy set F over the universe U it implies the existence of a corresponding possibility distribution with Poss(X = u) = $\mu_F(u)$. This observation explains why these two concepts are used interchangeably throughout this paper.

Note also that possibility distributions subsume the conventional and set-value representation, since

- a single-valued data item x corresponds to a distribution which has one element x with the possibility 1, i.e., x = $\{1.0/x\}$;
- a set-valued data item $\{x1, x2, x3\}$ corresponds to a possibility distribution which has elements with possibilities of 1.0, i.e. $\{x1, x2, x3\} = \{1.0/x1, 1.0/x2, 1.0/x3\}$.

The notion of α-sets [23] allows us to get from fuzzy to crisp sets. This is useful, for example, if we want to exhibit an element u $\in$ U that typically belongs to a fuzzy set F. In other words, to make a decision which is of binary type, we demand that the membership value of each resulting element is greater than some threshold $\alpha \in$ (0,1].

Definition 12 *Given a fuzzy set F over U. Then the* α*-level set of F, denoted by* F_α*, is defined by*

$$F_\alpha = \{u \in U \mid \mu_F(u) \geq \alpha\}. \tag{10}$$

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

$$F = \sum_\alpha \alpha F_\alpha \tag{11}$$

where αF_α is the product of a scalar α with the set F_α and $\sum_\alpha$ is the union with α ranging from 0 to 1.

Furthermore, the extension principle introduced by Zadeh [24] allows arithmetic operations based on numeric values to be extended to apply to possibility distributions.

Definition 13 *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 *with* Π_x *and* Π_y *over U by the following:*

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

where $\cap$ *is the minimum operator.*

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 since the minimum of two real numbers taken from [0,1] is well-defined and results in a real number again from the [0,1] interval. In the course of this paper, we will make extensive use of this definition, especially for defining generalized versions of the classical scalar aggregates.

5 EXTENDING THE RELATIONAL DATA MODEL

5.1 The Possibilistic Relational Data Model

Various attempts toward enhancing the relational database model by fuzzy extensions can be found in the literature [3, 14, 26, 18]. This section reviews the basics underlying most of these models and then describes our approach of enhancing the relational model by means of fuzzy set theory, which results in the possibilistic relational model [19]. The remainder of this work is based on the here discussed possibilistic relational model.

The concept of a fuzzy relation has been defined based on the notion of a fuzzy set. A fuzzy relation can be considered a generalization of a fuzzy set, i.e. a fuzzy subset of the Cartesian product of some universes of discourse.

Definition 14 *Let U be the Cartesian product of n universes of discourse* $U_1, ..., U_n$, *i.e.* $U = U_1$ x U_2 x ... x U_n. *Then an n-ary* **fuzzy relation** *r in U is a relation which is characterized by a n-variate membership function ranging over U, i.e.,*

$$\mu_r : U \rightarrow [0, 1] \tag{12}$$

Since the traditional relational data model is based on the foundation of set and relation theory, the proposal to adopt the concept of a fuzzy relation from fuzzy set theory as given in definition 14 for the enhanced data model has been made by several researchers [24, 27]. A n-ary fuzzy relation r over the relation schema R($A_1, A_2, ..., A_n$) as defined in definition 14 corresponds to a fuzzy subset of U_1 x U_2 x ... x U_n where U_i is the domain of A_i for all i. A tuple t_j of the fuzzy relation r can thus be expressed as

$$t_j = < u_{j1}, u_{j2}, ..., u_{jn}, \mu_r(u_{j1}, u_{j2}, ..., u_{jn}) >$$

Consequently, the relation r is captured by a tableau of the form in figure 4. It is important

A_1	A_2	...	A_n	μ_r
u_{11}	u_{12}	...	u_{1n}	$\mu_r(u_{11}, u_{12}, ..., u_{1n})$
...	...	...	...	...
u_{j1}	u_{j2}	...	u_{jn}	$\mu_r(u_{j1}, u_{j2}, ..., u_{jn})$
...	...	...	...	...

Figure 4: A fuzzy relation r.

to note that in a relational data model that can support imprecise information, it is necessary to accommodate two types of impreciseness, namely, the impreciseness in data values and impreciseness in the association among data values. The second type of impreciseness is, in general, expressed by the degree of membership of a tuple in a relation. As an example the membership value $\mu_{Good-at}$ of the tuples in the relation Good-at(Person,Skill) depicted in figure 5 expresses the degree to which a tuple belongs to the relation. Or, in other words, to what degree the relation holds. Note that this type of impreciseness has been modeled by the just described type of fuzzy relation. The first type of impreciseness refers to the

Name	Skill	$\mu_{Good-at}$
Mary	Dancing	1.0
Fred	Skiing	0.8

Figure 5: The fuzzy relation Good-at.

impreciseness of a data value, e.g. one may know that John is *old* but not his exact Age. This suggests a different approach for the extension of the relational model. Recall, that in general, the relational model consists of a set of relations comprised of tuples t_j for j = 1, ..., m of the form $< u_{j1}, u_{j2}, ..., u_{jn} >$, where each of these data values u_{ji} is selected from a given fixed domain, U_i. Thus, in the traditional data model each of these data values u_{ji} is a single value from a domain.

It is proposed to extend the set of possible data values to take different forms besides being constants. The data values for the possibilistic relational model [19] are extended to be (1) a single scalar, (2) a single number, (3) a set of scalars, (4) a set of numbers, (5) a possibilistic distribution of scalar domain values, (6) a possibilistic distribution of number domain values, (7) a real number from [0,1], and (8) a designated null value. It has already been pointed out that these eight possible data value types can be described by some form of a possibility distribution. This proposal of eight different data types is close

to the approach of Zemankova and Kandel [26]. Most other approaches in the literature restrict their models to a subset of the above, e.g. Buckles and Petry [3] and Oezsoyoglu, Oezsoyoglu, and Matos [12] allow only the data types (1) to (4), Umano [21] permits (1) to (4) and (7) and (8), and many others use only (7) besides (1) and (2), e.g. Zvieli [27] and Raju and Majumdar [15]. Now the following can be defined.

Definition 15 *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 the relation schema $R(A_1, A_2, ..., A_n)$ 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 $P(U_i)$ denote the collection of all possibility distributions on a universe of discourse U_i.

How can the possibilistic extension of the concept of a relation be described in tableau format? Let U_1, U_2, ..., U_n be again the universes of discourse upon which the possibilistic relation r is defined. Let $\Pi(A_i)$ be a possibility distribution of the attribute A_i defined on the universe U_i for all i. Then a tuple t_j of r has the form

$$t_j = <\Pi_j(A_1), \Pi_j(A_2), ..., \Pi_j(A_n)>.$$

The relation r can thus be represented by a tableau with n columns as shown in figure 6. Since this work is concerned with the aggregate evaluation on possibilistic data, we are primarily interested in the imprecision in the data and not in the imprecision of the association among entities. Consequently, we will limit our discussion to a possibilistic relational data model which consists of relations as defined in definition 15.

Definition 16 *A* **possibilistic relational database** *consists of a set of attribute names A_i, a set of corresponding domains U_i, and a collection of possibilistic relations r_i, i = 1,2, ..., m, as defined in definition 15.*

A_1	A_2	...	A_n
$\Pi_1(A_1)$	$\Pi_1(A_2)$	...	$\Pi_1(A_n)$
$\Pi_2(A_1)$	$\Pi_2(A_2)$	...	$\Pi_2(A_n)$
...	...	...	...
$\Pi_j(A_1)$	$\Pi_j(A_2)$	...	$\Pi_j(A_n)$
...	...	...	...

Figure 6: A possibilistic relation r.

Thus, the possibilistic relational model is characterized by a representation which allows for data values which can be modeled by possibility distributions. This includes, for example, multiple values (e.g. $\{23, 24, 25\}$), possibility distributions (e.g. $\{0.7/130, 0.8/135\}$), linguistic terms as labels for fuzzy sets (e.g. $young, about - 20, light$) or single values (e.g. 140) as data items. Note that, for example, the fuzzy set *light* could be represented as $\{1.0/100, 0.9/110, 0.7/120\}$. An example of a possibilistic relation is depicted in figure 7.

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 7: The possibilistic relation person

5.2 Relational Algebra for the Possibilistic Relational Model

Several suggestions can be found in the literature on how to extend the relational algebra operations to deal with possibilistic data [4, 14, 22, 26, 27, 18]. Important issues among

others are how to compare two possibility distributions and how to measure their similarity. We will limit our discussion here to the SELECT operation. For the other relational algebra operations see [15]. The SELECT operation is an extension of the SELECT operation described in definition 5. Now, instead of demanding the exact match between two values, an approximate match can be specified. An example for such a query is "Find all professors who are *old*" where the meaning of *old* has been defined as a fuzzy set on the domain Age as, for instance, depicted in figure 3.

Definition 17 *The syntax of the select operation is*

$$SELECT(r\ WHERE\ \ r.A_i\ is\ F) \tag{13}$$

where F refers to a fuzzy set defined over the domain of the attribute A_i.

The query can be evaluated by measuring the agreement of each tuple in the relation r with the fuzzy set F. This agreement, referred to as possibility measure by [14], is defined by

$$Poss(t[A_i]\ is\ F) = \max_{u \in A_i} \min(\pi_{A_i}(u), \mu_F(u)) \tag{14}$$

for all $u \in$ the domain U_i of A_i.

The result of a selection operation is a set of tuples, each associated with a 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 which match the selection criteria at least to that degree α. This corresponds to the notion of an α-level set as presented in definition 12.

Lemma 1 *Note that if the data is crisp, e.g., $t[A_i] = u = \{1.0/u\}$, then equation 14 simplifies to the following possibility measure for each individual tuple t of r:*

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

On the other hand, if we are dealing with possibilistic data but a crisp selection (as in definition 5), then equation 14 defaults to

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

with some $u \in U$.

A selection condition comparing two attribute values, which have been imprecisely specified, is evaluated similarly.

Definition 18 *Given the attributes* A_i *and* A_j *of relation* r. *The syntax for the query to find all tuples with matching attribute values for* A_i *and* A_j *is*

$$SELECT(r\ WHERE\ r.A_i = r.A_j) \tag{15}$$

The query can be evaluated by measuring the agreement of the two attribute values for each tuple in the relation.

$$Poss(t[A_i] = t[A_j]) = \max_{u \in dom(A_i) \cup dom(A_j)} \min(\pi_{A_i}(u), \pi_{A_j}(u))). \tag{16}$$

This evaluation could be extended to also incorporate the similarity between domain values [19]. However, in order to keep the discussion simple this is neglected here. Also, additional modifiers could be applied to extend these queries. An example is the query "Find all professors who are *very old*" where *very* is a modifier for the fuzzy set *old* [26].

6 EXTENDING FRQLs WITH AGGREGATES

In this section, it is investigated how the aggregate operators as presented in section 3 can be redefined to cope with the possibilistic representation of data. Note that the introduction of possibilistic information influences the evaluation of aggregates at different levels, namely,

- 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 is only concerned with aggregate functions, whereas the others have to consider both, scalar aggregates and aggregate functions.

An important goal of this research in either of the above cases is to base the generalizations of the aggregate evaluation on two principles [3]:

- **consistency:** the generalized operations should default to (be consistent with) the crisp operations for conventional data; and
- **completeness:** the operations should be well-defined for the fuzzy case.

These requirements state that the generalized versions of aggregates are supposed to be natural extensions of their crisp counterparts. The consistency requirement guarantees, at least to some degree, that the proposed extensions are sensible, since if they would not default to the crisp case, then they would obviously not capture the original meaning. It is conceptually straightforward to verify whether the consistency requirement holds. This is done by evaluating the respective extended definition for aggregate evaluation on crisp data. The same is not true for the completeness requirement. Moreover, it appears that only an empirical evaluation can determine whether the proposed operations are generally acceptable.

6.1 Aggregate Evaluation of Vague Queries on Crisp Data

In the following we describe how the aggregate operators as presented in section 3 can be extended to cope with vague queries/fuzzy predicates on crisp data. Examples of the types of queries considered in this section are "What is the smallest salary of *old* professors?" or "What is the average of the *high* salaries of all professors?". More general, the queries have the form

$$f((A_i)(SELECT(r\ WHERE\ \ \mathbf{r.A_j\ is\ F})))\qquad(17)$$

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 latter of the two example queries listed above indicates. Recall that F could actually be a more complicated expression with fuzzy modifiers or a conjunction/disjunction of several fuzzy sets. Without the loss of generality, this discussion is confined to F being a simple fuzzy set over the domain of the attribute A_j.

According to definition 17, or more precisely 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 of truth* with which the proposition holds, i.e. to what degree the tuple matches the selection criteria. This 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 choice of the level of acceptance, denoted by α. Recall, the larger α is, i.e. the higher the required degree of matching, the lower the number of elements able to satisfy α will be. In the following, we propose an extended version of an α-level set, called an α-level relation, and then give the definition of a scalar aggregate based on this new concept.

Definition 19 *Let r be a relation defined over the relation schema $R(A_i, A_j, ...)$. Let F be a fuzzy set over the domain of A_j denoted by μ_F. Let $\alpha \in [0,1]$. Recall that an $\alpha-$level set F_α as defined in definition 12 corresponds to all values which are at least with certainty α in F. Thus, F_α contains all $x \in r[A_j]$ with $\mu_F(x) \geq \alpha$. Let the α-level relation $A_j^r(\alpha)$ be defined as follows:*

$$A_j^r(\alpha) = \{t \mid t \in r \wedge (t[A_j] \in F_\alpha)\} = \{t \mid t \in r \wedge \mu_F(t[A_j]) \geq \alpha\} \tag{18}$$

Then for a given $\alpha \in [0,1]$ all tuples which fulfill the proposition "$r.A_j$ is F" at least to the degree α are collected in $A_j^r(\alpha)$. Thus, evaluate the aggregate f on $A_j^r(\alpha)$ for all α and associate α with the result. More precisely, the semantics of a query of the form as indicated

by equation 17 are defined to be:

$$f((A_i)(SELECT(r\ WHERE\ \ r.A_j\ is\ F))) = \{\mu_f(y)/y \mid \mu_f(y) = sup_\alpha\{f((A_i)(A_j^r(\alpha))) = y\}\}. \tag{19}$$

The fuzziness of F induces a fuzzy set of possible answers instead of one value. Again, similar to definition 17, one may be only interested in levels of acceptance above a certain threshold, and thus could discard the results for smaller α. The following example is given to demonstrate the above definition.

Example 5 *Let Prof be the relation defined in figure 1. Let Prestigious be a fuzzy set defined on the set of different positions as represented in figure 8. Then the query "What*

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

Figure 8: The fuzzy set Prestigious

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 19). 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 18. Then applying the average aggregate to the tuples in $Position^{Prof}(1.0)$ results in

avg((Salary) ($Position^{Prof}(1.0)$)) = avg(4500, 4000) = 4250.

For α = 0.8, $Position^{Prof}(\alpha) = \{t2, t3, t4, t5\}$. Then avg((Salary) ($Position^{Prof}(0.8)$)) = avg(4500, 4000, 2500, 3500) = 3625.

For $\alpha = 0.5$, $Position^{Prof}(0.5) = \{t1, t2, t3, t4, t5\}$. Then avg((Salary) ($Position^{Prof}(0.5)$)) = avg(3500, 4500, 4000, 2500, 3500) = 3600.

Thus, the result of the query is the fuzzy set $\{1.0/4250, 0.8/3625, 0.5/3600\}$. *It has to be remarked that the expression avg((Sal)($Position^{Prof}(\alpha)$)) evaluates to 4250 for all $\alpha \in$ [1.0,0.8), to 3625 for all $\alpha \in$ [0.8,0.5), to 3600 for all $\alpha \leq 0.5$. As indicated in equation 19, we have already taken the supremum of all α's for a given result y by concentrating on distinct α values being used as membership values in the fuzzy set F. This is an obvious step, since otherwise we would have to calculate infinitely many redundant α-level relations.*

Clearly, the threshold level α determines which values are taken into consideration for the evaluation. The smaller α is the more elements are going to be included into the evaluation. Including more elements into the calculation of an α-level set has the following consequences.

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 [13]. It may be of interest to summarize the result of such a query in a more concise way. A simple approach may be to give an α value which one considers as having a sufficient matching degree; then the result of applying the aggregate to $A^r(\alpha)$ may be returned, which is one single value. Finally note that 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.

The requirement for consistency as stressed at the beginning of this section is met, since the definition of the aggregate operations as such has not been altered at all. The approach

presented accounts for the possibilistic relational data model's ability to specify imprecise queries and its impact on the aggregate evaluation process.

6.2 Scalar Aggregates for Possibilistic Data

This section discusses how to evaluate precisely specified queries on possibilistic data found in the possibilistic relational data model. Thus, the queries considered correspond to the ones described in definition 6. Obvious problems arise from the fact that possibility distributions are now allowed as attribute values instead of simple constants. For example, the number of values per attribute (column) will in general no longer equal the number of tuples per relation.

6.2.1 Generalized Count Aggregate

The count aggregate, defined in definition 7, returns the number of values for a given attribute. It is possible to directly adopt this definition for the possibilistic case, if one interprets it as counting the number of tuples and not the number of existing $\mu(ui)/ui$ pairs. This is referred to as $fcount1$. An alternative is the use of the sigma-count operation [24, 10], which is defined as follows.

Definition 20 *Given a fuzzy subset F of* $U = \{u1, u2, ..., un\}$ *with* $F = \{\mu(u1)/u1, \mu(u2)/u2, ..., \mu(un)/un\}$ *with* $\mu(ui)$ *being the grade of membership of ui in F. Then, the cardinality of F, called sigma-count, is the arithmetic sum of the grades of memberships in F. Thus,*

$$sigma\text{-}count(F) = \sum_{\forall i} \mu(ui)$$

It is proposed here to use the sigma-count as count aggregate. This definition of a generalized count aggregate is referred to as $fcount2$. The result of a $fcount2$ operation is a real number, but it is understood that the result may be rounded, if need be. The $fcount2$ operation does not exhibit all features of the conventional count operation, for example,

it does necessarily return the same value for the different attributes of a relation. If this characteristic is required, then the designer will choose the fcount1 operator as generalized version of the count operator over the fcount2 aggregate. The $fcount2$ operator has been proven useful for defining the generalized average operator (see section 6.2.5).

Example 6 *The $fcount1$ for the Weight attribute of the person relation in figure 7 is 4, since there are four tuples. Whereas, the $fcount2$ aggregate results in*

$$fcount2((Weight)(person))$$
$$= \sum_{\forall i,j} \mu(u_{ij})$$
$$= (1.0) + (1.0 + 0.9 + 0.7)\ (0.6 + 1.0) + (1.0) = 6.2$$

The consistency requirement demands that the definition of the generalized count aggregate operation, $fcount2$, defaults to the classical count definition for attributes with singletons.

Lemma 3 *The fcount2 aggregate is consistent.*

Proof: Given a relation r (possibilistic or conventional) defined on the relation schema R(..,A,...). Let A be defined on the domain U = $\{u_1, ..., u_n\}$ and let the relation r take crisp values on the attribute A. The relation r consists of tuples t_i with 1 $\leq$ i $\leq$ m. In the crisp case, each tuple t_i of r takes crisp values for the attribute A, i.e. $(t_i[A] \in U)(\forall i)$. Thus, t_i has but one value u_{ij} with possibility 1.0 for the attribute A , i.e. $t_i[A] = \mu_A(u_{ij})/u_{ij} = 1.0/u_{ij}$, which is another notation for u_{ij}. Thus, altogether,

$$fcount2((A)(r)) = \sum_{i=1}^{m} \sum_{j=1}^{n} \mu_A(u_{ij}) = \sum_{i=1}^{m} 1.0 = m.$$

Recall that m stands for the number of tuples in the relation r, and thus fcount2 defaults to count.

q.e.d.

6.2.2 Generalized Sum Aggregate

The sum aggregate is, like some of the other aggregates, an arithmetic operation, and thus is only defined for numeric domains. Necessary characteristics of the extended sum aggregate definition are the commutativity and associativity, since the relational data model does not place any order on the tuples of a relation but considers them as a 'set of tuples' [6].

The sum aggregate, here termed $fsum$, is defined based on the definition 21, which in turn is an application of definition 13.

Definition 21 *Let + be the binary plus operation. Let Π_x and Π_y be two possibility distributions defined on the universe of discourse U. Then, the sum of possibility distributions is defined as*

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

This extension of the binary plus operation to the addition of possibility distributions is well-defined, since by assumption the operation $(u1 + u2)$ is well-defined for u1, u2 $\in$ U and since the minimum of two real numbers taken from [0,1] is well-defined and results in a real number again from the [0,1] interval.

Definition 22 *Given a possibilistic relation r defined on the relation schema R(..,A,...). Let A be an attribute defined on the domain $U = \{u_1, ..., u_n\}$ and let r consist of tuples t_i with $1 \leq i \leq m$, and $t_i[A]$ has the form $\{\mu_i(u_1)/u_1, ..., \mu_i(u_j)/u_j, ..., \mu_i(u_n)/u_n\}$. The fsum aggregate of the attribute A of a relation r is defined based on the extended addition operation described in definition 21. It is:*

$$fsum((A)(r))$$

$= \{u/y \mid ((\mu_1(u_{k1})/u_{k1} \in t_1[A]) \wedge (\mu_2(u_{k2})/u_{k2} \in t_2[A]) .... \wedge (\mu_m(u_{km})/u_{km} \in t_m[A])$
$\wedge (y = \sum_{ki=k1}^{km} u_{ki}) \wedge (u = \min_{i=1}^{m} \mu_i(u_{ki}))) \ (\forall k1, ..., km : 1 \leq k1, ..., km, \leq n)$
$\wedge ((t_i \in r) \wedge (t_i[A] = \{ \mu_i(u_1)/u_1, ..., \mu_i(u_n)/u_n\}) \ (\forall i))\}$

It is relatively easy 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. The fsum aggregation of the Weight attribute of the person relation of figure 7 is calculated in the following example.

Example 7 *The fsum aggregation of the Weight of the relation person of figure 7 is*

$$fsum((Weight)(person))$$
$$= \{1.0/130\} + \{1.0/100, 0.9/110, 0.7/120\} + \{0.6/120, 1.0/130\} + \{1.0/110\}$$
$$= \{1.0/(100+130), 0.9/(110+130), 0.7/(120+130)\} + \{0.6/230, 1.0/240\}$$
$$= \{1.0/230, 0.9/240, 0.7/250\} + \{0.6/230, 1.0/240\}$$
$$= \{0.6/460, 0.6/470, 0.6/480, 1.0/470, 0.9/480, 0.7/490\}$$
$$= \{0.6/460, 1.0/470, 0.9/480, 0.7/490\}$$

This result states that the sum of all values is 470 with the possibility of 1.0, and that some of the values close to 470 are also possible results.

Again, it can be shown that the *fsum* aggregate defaults to the conventional sum aggregate in the case of crisp data values.

Lemma 4 *The fsum operation is consistent.*

Proof: Given a relation r (possibilistic or conventional) defined on the relation schema R(..,A,...). Let A, defined on the domain U = $\{u_1, ..., u_n\}$, be a crisp attribute. Let r consist of tuples t_i with 1 $\leq$ i $\leq$ m. In the crisp case, each tuple t_i of r takes crisp values for the attribute A, i.e. $(t_i[A] \in U)$ $(\forall i)$. In other words, all t_i[A] have the form $\mu_i(u_j)/u_j$ with $\mu_i(u_j) = 1.0$. So, $t_i[A] = u_j$ for some j. Thus, altogether,

$$\text{fsum}((\text{A})(\text{r}))$$

$$= \{u/y \mid ((\mu_1(u_{k1})/u_{k1} = t_1[A]) \wedge (\mu_2(u_{k2})/u_{k2} = t_2[A]) \ldots . \wedge (\mu_m(u_{km})/u_{km} = t_m[A])$$

$$\wedge (y = \sum_{ki=k1}^{km} u_{ki}) \wedge (u = \min_{i=1}^{m} \mu_i(u_{ki}))$$

$$\wedge \mu_1(u_{k1}) = 1.0 \wedge \mu_2(u_{k2}) = 1.0 \wedge \mu_m(u_{km}) = 1.0)$$

$$(for\ some\ k1, ..., km \in \{1, ..., n\}.)\}$$

$$= \{u/y \mid ((u_{k1} = t_1[A] \wedge u_{k2} = t_2[A] \wedge u_{km} = t_m[A]$$

$$\wedge (y = \sum_{ki=k1}^{km} u_{ki}) \wedge (u = \min_{i=1}^{m} 1.0))$$

$$(for\ some\ k1, ..., km \in \{1, ..., n\}.)\}$$

$$= 1.0/\sum_{i=1}^{m} t_i[A]$$

$$= \sum_{i=1}^{m} t_i[A]$$

$$= \text{sum}((\text{A})(\text{r}))$$

q.e.d.

The consistency results from the notion of the extension principle, which has the goal of consistently extending conventional arithmetic operations to apply to possibility distributions.

6.2.3 Generalized Max Aggregate

The max aggregate is only defined for numeric domains. The extension of the max aggregate to deal with possibility distributions, here called $fmax$, is again based on definition 13. Consequently, commutativity and associativity of the operation are given.

Definition 23 *Given a possibilistic relation r defined on the relation schema R(..,A,...) with A defined on the domain* $U = \{u_1, ..., u_n\}$. *Let r consist of tuples* t_i *with* $1 \leq i \leq m$. $t_i[A]$ *has the form* $\{\mu_i(u_1)/u_1, \ldots., \mu_i(u_j)/u_j, \ldots., \mu_i(u_n)/u_n\}$ *for all i. Let max be the maximum operation. The fmax aggregate of the attribute A of the relation r is defined based on the extension principle as described in definition 13:*

$$fmax((a)(r))$$

$$= \{u/y \mid ((\mu_1(u_{k1})/u_{k1} \in t_1[A]) \wedge (\mu_2(u_{k2})/u_{k2} \in t_2[A]) \ldots . \wedge (\mu_m(u_{km})/u_{km} \in t_m[A])$$
$$\wedge (y = \max_{ki=k1}^{km} u_{ki}) \wedge (u = \min_{i=1}^{m} \mu_i(u_{ki}))) \ (\forall k1, ..., km : 1 \leq k1, ..., km, \leq n)$$
$$\wedge((t_i \in r) \wedge (t_i[A] = \{ \mu_i(u_1)/u_1, ..., \mu_i(u_n)/u_n\}) \ (\forall i))\}$$

This generalization of the maximum aggregate is well-defined, since by assumption the max operation is well-defined for elements $\in$ U and since the minimum of two real numbers taken from [0,1] is well-defined and results in a real number again from the interval [0,1]. The result of the $fmax$ aggregate is in general a possibility distribution. An example of the $fmax$ operation is given beneath.

Example 8 *The fmax of the Weight attribute of relation person of figure 7 is:*

$$fmax((Weight)(person))$$
$$= fmax(\{1.0/130\} + \{1.0/100, 0.9/110, 0.7/120\} + \{0.6/120, 1.0/130\} + \{1.0/110\})$$
$$= fmax(\{1.0/max(100, 130), 0.9/max(110, 130),$$
$$0.7/max(120, 130)\}, \{0.6/max(120, 110), 1.0/max(130, 110)\} \ )$$
$$= fmax(\{1.0/130, 0.9/130, 0.7/130\}, \{0.6/120, 1.0/130\})$$
$$= \{0.6/130, 0.6/130, 0.6/130, 1.0/130, 0.9/130, 0.7/130\} = \{1.0/130\} = 130$$

This turns out to be a very realistic result, since 130 is indeed the maximum value.

Note, again, that the $fmax$ aggregate defaults to the conventional max aggregate in the case of crisp data values.

Lemma 5 *The fmax operation is consistent.*

Proof: This can be shown in a manner equivalent to the one used in the proof of lemma 4 in the previous section by simply replacing sum by max.

q.e.d.

6.2.4 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 23 by min. For a more thorough discussion of the default, etc., consult the previous section.

Example 9 *The fmin of the Weight attribute of relation person of figure 7 is:*

$$fmin((Weight)(person)) = \{1.0/100, 0.9/110\}$$

The result of 100 or 110 for the minimum is very intuitive, since both are among the lowest values of the Weight attribute appearing in the person relation.

6.2.5 Generalized Avg Aggregate

We present a framework for two possible routes one could take in generalizing the definition of the average aggregate. It is ultimately up to the designer to choose the appropriate one for the application at hand. One could either define the generalized average aggregate in terms of the quotient of the generalized sum and count aggregates, or alternatively, one could search for an independent definition. The first approach which results in the average operator $favg1$ is presented beneath, whereas the presentation of the second one is deferred until later in this section.

Definition 24 *Let fcount1 and fsum be the generalized aggregates as described in sections 6.2.1 and 6.2.2. Then, the favg1 aggregate can be defined as*

$$favg1((A)(r)) = \frac{fsum((A)(r))}{fcount1((A)(r))}$$

Since the $fsum$ operation results in a possibility distribution and the $fcount$ operation in a real number, the average aggregation is essentially a quotient of a possibility distribution and a real number. Hence, the result is well-defined; in fact, it produces a possibility distribution when calculating the division operation in accordance with the strategy presented

in definition 13. As an example consider the average of all values of the Weight attribute of relation person depicted in figure 7.

Example 10 *The fsum of the Weight attribute was fsum((Weight)(person))* $= \{0.6/460, 1.0/470, 0.9/480, 0.7/490\}$. *Also, fcount1((Weight)(person)) = 4. Thus,*

$$favg1((Weight)(person))$$

$$= \frac{fsum((Weight)(person))}{fcount1((Weight)(person))}$$

$$= \frac{\{0.6/460,1.0/470,0.9/480,0.7/490\}}{4}$$

$$= \{0.6/115, 1.0/117.5, 0.9/120, 0.7/122.5\}$$

Again, an average of around 120 is very realistic.

As a matter of course, this generalized operation will default to its conventional counterpart in the crisp case.

Lemma 6 *The favg1 operation is consistent.*

Proof: It has previously been shown, that for crisp data $fsum$ and $fcount1$ default to *sum* and *count*, respectively. Thus, for crisp data,

$$favg1((A)(r)) = \frac{fsum((A)(r))}{fcount1((A)(r))} = \frac{sum((A)(r))}{count((A)(r))}.$$

This is the classical definition of the average aggregate as given in definition 7.

q.e.d.

An alternative to the approach just described is to make use of the possibilistic expected value, PEV a concept introduced by Zemankova and Kandel [26]. Zemankova and Kandel propose to use the PEV value as a default value in place of a null value in a query evaluation.

Definition 25 *Let A be an attribute of a numeric domain of relation R. Let $\pi(u_i)$ be the possibility distribution for value ui, and n the number of $\pi(u_i)/u_i$ pairs for the attribute A. Let * stand for multiplication. The PEV for attribute A is defined by*

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

This idea will be adopted here. We propose to define the average aggregate $favg2$ in terms of the PEV operation. In fact, the definition of the $favg2$ aggregate corresponds to the concept of a PEV except for the replacement of the denominator n by $fcount2$.

Definition 26 *Given a possibilistic relation r defined on the relation schema R(..,A,...). Let A be defined on the domain $U = \{u_1, ..., u_n\}$. Let r consist of tuples t_i with $1 \leq i \leq m$. Let $t_i[A] = \{\mu_i(u_1)/u_1, ..., \mu_i(u_n)/u_n\}$ for all i. Now, the favg2 operator is defined to be*

$$favg2((A)(r)) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} \mu_i(u_j)*u_j}{fcount2((A)(r))}.$$

Note, that the $favg2$ operator results in a real number, whereas $favg1$ results in a possibility distribution. This is an important distinction between these alternatives, based on which a designer may make a choice. It is understood that favg2 may be rounded, if need be. The consistency of the $favg2$ aggregate can again be shown.

Lemma 7 *The favg2 aggregate is consistent.*

Proof: Given a relation r defined on the relation schema R(..,A,...). Let the attribute A, which is defined on the domain U = $\{u_1, ..., u_n\}$, be a crisp attribute. Let r consist of tuples t_i with 1 ≤ i ≤ m. Since A is crisp, we have $t_i[A] = \mu_i(u_{i_j})/u_{i_j} = 1.0/u_{i_j} = u_{i_j}$ for all i $\in \{1, ..., m\}$ and for some $u_{i_j} \in$ U. Then,

$$favg2((A)(r))$$

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

$$= \frac{\sum_{i=1}^{m} \mu_i(u_{i_j}) * u_{i_j}}{fcount2((A)(r))}$$

$$= \frac{\sum_{i=1}^{m} 1.0 * u_{i_j}}{fcount2((A)(r))}$$

$$= \frac{\sum_{i=1}^{m} t_i[A]}{count((A)(r))}$$

$$= \frac{sum((A)(r))}{count((A)(r))}$$

$$= avg((A)(r))$$

q.e.d.

The $favg2$ operator is demonstrated again on the Weight attribute of the relation person of figure 7.

Example 11 *The favg2 aggregate on attribute Weight result is calculated in the following. Recall that fcount2((Weight)(person)) = 6.2.*

$$favg2((Weight)(person))$$
$$= favg2(\{1.0/100, 0.9/110, 0.7/120\}, \{1.0/130\},$$
$$\{0.6/120, 1.0/130\}, \{1.0/110\})$$
$$= (1.0 * 100 + 0.9 * 110 + 0.7 * 120 + 1.0 * 130 + 0.6 * 120$$
$$+1.0 * 130 + 1.0 * 110)/6.2$$
$$= (100 + 99 + 84 + 130 + 72 + 130 + 110)/6.2$$
$$= 725/6.2 = 116.9$$

The favg2 results in a satisfactory value, and since a real number is compacter than a possibilistic distribution, the designer may prefer the favg2 over the favg1 operator.

Finally, the conventional *any* operator can be directly adopted from definition 7 since it tests whether there is a tuple in the relation or not, and thus does not concern the actual content of the relation.

6.3 Aggregate Functions for Possibilistic Data

As was outlined in section 3.2, aggregate functions are in large based on the evaluation of scalar aggregates. This also holds true for aggregate functions in the possibilistic relational data base. 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 6.3.2.

6.3.1 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 9 by translating the individual operations, such as Cartesian product, selection by WHERE clause, etc., to their corresponding fuzzy counterparts of the respective FRQL. Thus, it may effect step 1 of the evaluation process described in definition 9. This is so since 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 6.2. The following example demonstrates the procedure of evaluating an aggregate function over possibilistic data by partitioning it on precise data.

Example 12 *Let person be the relation given in figure 7. 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 by each sex?" is formally expressed by favg2((Weight)(person) BY SEX) if we choose the favg2 as the generalized average aggregate operator. This can be evaluated as follows:*

1. *Step one of the evaluation process as given in definition 9 is not needed, since the person relation is a base relation.*

2. *First, find the attribute values to partition on by*

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

Thus, by equation (6) 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. Since we have chosen favg2 as the average operator for the aggregate function, we stick with this choice for the application of the scalar aggregate.*

- $favg2((Weight)(P^{Sex}_{person}(male))) = favg2(\ 130, \{0.6/120, 1.0/130\}) = (130 + 72 + 130)/2.6 = 127.7.$
- $favg2((Weight)(P^{Sex}_{person}(female))) = favg2(\ 110, \{1.0/100, 0.9/110, 0.7/120\}) = (100 + 99 + 84 + 110)/3.6 = 109.1.$

4. *Associate the result gained for each partition with the value on which that respective partition was based. This now may be represented by a relation with two attributes, namely, the partition attribute Sex and the result attribute Avg-Weight. The result relation, called avg-weight-by-sex, is depicted in figure 9.*

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

Figure 9: The avg-weight-by-sex relation

The average operator, $favg2$, which was applied in the previous example results in a real number. The application of an aggregate function on a possibilistic relation does not have to be a real number. For instance, if we were to chose $favg1$ over $favg2$, then the values of the Avg-Weight attribute of the relation avg-weight-by-sex relation in figure 9 would be possibilistic.

6.3.2 Partitioning on Possibilistic Data

Now, let us consider the case when the partitioning process is based on an attribute containing possibilistic data. For simplicity reasons, we assume that the attribute over which the aggregate is to be evaluated is of crisp nature. 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 6.1. The following definition, a variation of definition 19, describes how the aggregate function $f((A_i)(r)BYA_j)$ is evaluated allowing possibilistic values for A_j while A_i consists of only crisp values. The problem of partitioning on possibilistic data leads to the introduction of a new concept, the α-level partition, which is defined in equation 21.

Definition 27 *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 6 has to be modified to accommodate for membership values. Thus, the partition function is now defined for all $a \in D$ by*

$$P_r^{A_j}(a) = \{t \mid (t \in r) \wedge (a \in D) \wedge (\mu_{t[A_j]}(a) > 0)\} \tag{20}$$

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

$$L_r^{A_j}(\alpha, a) = \{\ t \mid (t \in P_r^{A_j}(a)) \wedge (\mu_{t[A_j]}(a) \geq \alpha)\} \tag{21}$$

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 all associated α-level partitions $L_r^{A_j}(\alpha, a)$ for all α. More precisely,

$$f((A_i)(r)BY A_j)$$

$$= \{a \circ F \mid (a \in D) \wedge F = \{\mu_f(y)/y \mid \mu_f(y) = sup_\alpha\{f((A_i)(L_r^{A_j}(\alpha, a))) = y\}\}. \tag{22}$$

An example is shown 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 10: The Prof relation

Example 13 *Given the relation Prof depicted in figure 10. Note that the information provided in the relation about the positions of the professors is uncertain, i.e. possibilistic. Thus the query "What is the maximum salary of professors per position" which is formally expressed by fmax((Salary)(Prof) BY Position) has to be evaluated according to definition 27.*

1. *Again, step 1 of the evaluation process as described in definition 9 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 (6). Each tuple in a partition has associated a membership value indicating to what degree it participates in that partition:

- $P_{Prof}^{Position}(Full) =$
  $\{(Jack, 4500, \{1.0/Full\})\ with\ 1.0;$
  $(Julie, 4000, \{1.0/Full, 0.5/Associate\})\ with\ 1.0.\}$
- $P_{Prof}^{Position}(Associate) =$
  $\{(Tom, 3500, \{1.0/Assistant, 0.8/Associate\}) =\ with\ 0.8;$
  $(Julie, 4000, \{1.0/Full, 0.5/Associate\})\ with\ 0.5;$
  $(Mary, 2500, \{1.0/Associate, 0.7/Assistant\})\ with\ 1.0;$
  $(Frank, 3500, \{1.0/Associate\})\ with\ 1.0\}$
- $P_{Prof}^{Position}(Assistant) =$
  $\{(Tom, 3500, \{1.0/Assistant, 0.8/Associate\})\ with\ 1.0;$
  $(Mary, 2500, \{1.0/Associate, 0.7/Assistant\})\ with\ 0.7\}$

3. *Now, by equation 21, 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, apply the scalar aggregate operator, maximum, to all α-level partitions $L_{Prof}^{Position}(\alpha, a)$ associated with a given partition $P_{Prof}^{Position}(a)$:*

- $fmax((salary)(P_{Prof}^{Position}(Full))$:

- *for α = 1.0: fmax((Salary)($L_{Prof}^{Position}$(1.0, Full)) = max(4500, 4000) = 4500.*

- *fmax((salary)($P_{Prof}^{Position}$(Associate)) :*

  - *for α = 1.0: fmax((Salary)($L_{Prof}^{Position}$(1.0, Associate)) = max(2500, 3500) = 3500.*

  - *for α = 0.8: fmax((Salary)($L_{Prof}^{Position}$(0.8, Associate)) = max(2500, 3500) = 3500.*

  - *for α = 0.5: fmax((Salary)($L_{Prof}^{Position}$(0.5, Associate)) = max(2500, 3500, 4000) = 4000.*

- *fmax((salary)($P_{Prof}^{Position}$(Assistant)) :*

  - *for α = 1.0: fmax((Salary)($L_{Prof}^{Position}$(1.0, Assistant)) = max(3500) = 3500.*

  - *for α = 0.7: fmax((Salary)($L_{Prof}^{Position}$(0.7, Assistant)) = max(2500, 3500) = 3500.*

4. *Now associate the result of the scalar aggregate evaluation for each α-level partition with the respective level value 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. Associate this possibility distribution with the value the partition was based on. This results in a relation defined on a relation schema R consisting of two attributes, R(Position,Max-Salary) which is depicted in figure 11.*

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 27 the result of the third step of the aggregate evaluation process would be of the following form: e.g., $0.7/\{1.0/2500, 0.8/3500, 0.5/4000\}$. This can be evaluated by taking the minimum of the membership values, which in this example results in $\{0.7/2500, 0.7/3500, 0.5/4000\}$.

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

Figure 11: The max-sal-per-position relation

6.4 Vague Queries on Possibilistic Data

We conclude with a discussion on how to combine the aggregate evaluation approach dealing with vague queries (section 6.1) with the one proposed for handling possibilistic data (section 6.2 and section 6.3). At this point, we are favorable to simplifying the evaluation strategy as indicated at the end of section 6.1. 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 fulfill that selection criteria at least with the degree α. Hence, the result of a vague selection would be an α-level relation described in equation 18.

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 19 is modified by evaluating the vague selection in accordance with equation 14 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 over the threshold α are collected in the respective α-level relation, i.e., they are going to be further considered in the query evaluation process. In terms of the aggregate evaluation process outlined in definition 9, this means that the vague query part is dealt with in step 1 of the process. A non-base relation (an α-level relation

) is generated based on which the aggregate evaluation process can continue with steps 2, 3, and 4 as discussed in section 6.2 and 6.3.

7 CONCLUSIONS

The paper presents the possibilistic relational data model [19], which 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, we are able to handle vague queries. In short, the appropriate blending of fuzzy set and possibility theory with the relational database model has led to a true enhancement of the capabilities of existing database systems.

However, in order to make use of the possibilistic relational database model in real world applications, we find it essential to also develop suitable aggregate evaluation techniques. This is so since aggregates allow the formulation of queries that otherwise could not be specified. Hence, they are being supported by any of the existing relational database system, such as, System R [1] or Ingres [7].

The extensions of existing aggregate operators of conventional RQLs to their fuzzy counterparts have to fulfill two conditions. First, they have to be consistent, meaning they have to default to the crisp definitions when used on crisp data. Secondly, they have to be complete, i.e. return sensible and reasonable results.

In examining the approach of evaluating aggregates in the relational database system, we have found the distinction between scalar aggregates and aggregate functions a very important one. Both types of operators are essential, and thus we have developed a framework to cope with both of them.

First, we extended the definition of conventional aggregate operators to cope with vague queries. The major problem here is that the result of a vague query is commonly not a simple set of tuples, but a collection of pairs consisting of tuples and their associated

possibility measures indicating to what degree the respective tuple participates in that result. It turns out that we have to include this possibility measure into the evaluation of the aggregate operators in as much as it determines to what degree the tuple ought to contribute to the result of the evaluation. We have accomplished this by introducing the concept of an α-level relation.

Then, section 6.2 presents our approach of extending scalar aggregates to apply to possibilistic data found in the possibilistic relational data model. The evaluation of scalar aggregates for possibilistic data makes use of various concepts developed within the framework of fuzzy set and possibilistic theory, such as, the extension principle introduced by Zadeh [23], the sigma-count operation [24], and the concept of a possibilistic expected value, developed by Zemankova and Kandel [26] to cope with null values in relational databases. We were able to prove the consistency of the extensions of the aggregate operators. Furthermore, in order to demonstrate the intuitiveness of the proposed operators, we have shown the results of each of the operations on the same example relation. This work represents a sound framework for the remainder of our research, since scalar aggregates play an essential role in the evaluation of aggregate functions. This is true for the conventional as well as the possibilistic relational data model.

Finally, we show that the handling of aggregate functions in the possibilistic relational data model can be handled in a natural and intuitive manner. The case of partitioning on precise data defaults to a clean merge of the evaluation process of conventional aggregate functions (for which we have suggested four simple steps) with the approach of evaluating scalar aggregates on possibilistic data. When partitioning on possibilistic data, however, some adjustments have to be made. The reason for this is that we are no longer dealing with true partitions, since tuples can participate in more than one of these 'partitions'. We solve this problem by introducing the notion of an α-level partition. An α-level partition combines the concept of partitioning functions with the notion of α-level sets found in fuzzy

set theory. Finally, to base the partitioning as well as the actual aggregate evaluation on possibilistic data falls in place.

The integration of aggregates into FRQLs results in an increase in the expressive power of these relational languages. Since these operations are tightly coupled with the database systems, optimization techniques can be applied and thus one may be able to achieve this type of functional knowledge with satisfactory performance.

Possible extensions to this research are manifold. Future research should include the following issues.

- The development of new aggregates which may be suitable for the possibilistic relational model, even though they do not have a crisp counterpart. Some initial proposals have been made by Rundensteiner and Bic in [16];

- An investigation of the definition of aggregate operations for fuzzy data models which are not based on the notion of possibility distributions. For example, an extension of the aggregate concept to continuous domains, such as

  - interval representations, e.g., Weight[John] = [55.5,95.1]

  - functional descriptions, e.g., Weight[John] = light,

  where light is a continuous function on the real numbers between 100 and 200 [25]. It would be interesting to develop a common representation of the data to be found in a respective model, much like the possibilistic representation offers a common technique to represent all types of information to be captured in the possibilistic relational data model. The existence of diverse representations within the same model, such as, intervals, functions, etc., is bound to complicate the aggregate evaluation process considerably.

- The possibilistic relational database model deals with an approximate query, such as, "Find all *old* people", by retrieving a set of tuples and their associated degrees of truth in the proposition. The question how to address a modification operation, such as "Update the salaries of all *old* people" is still an open problem, since one could hardly update a value *to only some degree.*

- An implementation of the here proposed concepts of aggregates as add-ons to one of existing extended relational models would be desirable, since it would allow for empirical evaluations of the proposed aggregate evaluation methods.

References

[1] Astrahan, M. M., Blasgen, M. W., Chamberlin, D. D., and others. System R: Relational approach to database management. *ACM Trans. Database Systems* 1, 2 (1976), 97 - 137.

[2] Bandler, W. and Kohout, L. Fuzzy Power Sets and Fuzzy Implication operators. *Fuzzy Sets and Systems* 4, (1980), 13 - 30.

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

[4] Buckles, B. P., Petry, F. E., and Sachar, H. S. A Domain Calculus for Fuzzy Relational Databases. Tech. Rep. CSE TR-85-006, Dep. of Computer Science Engineering, Uni. of Texas, Arlington, (Nov. 1985).

[5] Codd, E. F. Extending the Relational Database Model to capture more meaning. *ACM Trans. on Database Systems* 4, 4 (Dec. 1979), 397 - 434.

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

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

[8] Date, C. J. *An Introduction to Database Systems.* $3^{r}d$ ed., Addison-Wesley, Reading, MA, 1981.

[9] Dubois, D., and Prade, H. The Treatment of Uncertainty in Knowledge-Based Systems Using Fuzzy Sets and Possibility Theory. *Int. Journal of Intelligent Systems* 3, 2 (Summer 1988), 141 - 165.

[10] Kandel, A. *Fuzzy Mathematical Techniques with Applications.* Addison-Wesley Pub. Co, Reading, Mass, 1986.

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

[12] 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.

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

[14] 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.

[15] 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.

[16] Rundensteiner, E. A., and Bic, L. Extending Fuzzy Relational Query Languages by Aggregates. *Proc. of NAFIPS'88*, San Francisco, CA, (June 1988), 201 - 205.

[17] Rundensteiner, E. A., Hawkes, L. W., and Bandler, W. A set-valued temporal knowledge representation for fuzzy temporal retrieval in ICAI. *Proc. of NAFIPS'87.* West Lafayette, Indiana, (May 1987), 37 - 65.

[18] Rundensteiner, E. A., Bandler, W., Kohout, L., and Hawkes, L. W. An investigation of fuzzy nearness measures. *Second IFSA Congress.* Meiji University, Tokyo, Japan, (July 1987).

[19] 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.

[20] Ullman, J. D. *Principles of Database Systems.* 2^{nd} ed., Computer Science Press, Rockville, MD, 1983.

[21] Umano, M. Freedom-0: A Fuzzy Database System. *Fuzzy Information and Decision Processes.* M. M. Gupta and E. Sachez, (eds), North-Holland Pub. Co., (1982), 339 - 347.

[22] Umano, M. Retrieval from fuzzy database by fuzzy relational algebra. *Knowledge Information Systems, Medical Applications.* (1983), 1 - 6.

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

[24] Zadeh, L. A computational approach to fuzzy quantifiers in natural languages. *Computers and Mathematics* 9, 1 (1983), Pergamon Press Ltd, Great Britain, 149 - 184.

[25] Zadeh, L. Fuzzy Logic. *Computer* 21, 4 (April 1988), 83 - 93.

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

[27] Zvieli, A. On Complete Fuzzy Relational Query languages. *Proc. of NAFIPS'86.* New Orleans, (1986), 704 - 726.

Contents