Fuzzy Analytical Queries: A New Approach to Flexible Fuzzy Queries

1st Sławomir Zadrożny
Systems Research Institute
Polish Academy of Sciences
Warsaw, Poland
Slawomir.Zadrozny@ibspan.waw.pl
https://orcid.org/0000-0002-6642-0927

2nd Janusz Kacprzyk
Systems Research Institute
Polish Academy of Sciences
Warsaw, Poland
Janusz.Kacprzyk@ibspan.waw.pl
https://orcid.org/0000-0003-4187-5877

***Abstract*—A new approach to the use of fuzzy terms in analytic functions which constitute a standard part of the SQL syntax is proposed. The motivation is that though extensions of the SQL queries including linguistic (fuzzy) terms have been widely used as they have made it possible to better and more directly represent complex requirements of a human user searching a relational database, notably for generating linguistic summaries, there has been little attention paid so far to the use of analytic functions in this context. Analytic functions can provide for an attractive way of analysing data in a relational database and may be possibly adopted as a tool to generate even more sophisticated linguistic summaries. In this paper we study a basic structure of standard (crisp) analytic functions use and point out some opportunities to extend it using linguistic terms. A starting point for our discussion is the mechanism of grouping rows in SQL as it is one of the most important components of the application of analytic functions.**

***Index Terms*—flexible fuzzy queries, analytic functions, grouping data, linguistic data summaries**

I. Introduction

Flexible fuzzy queries were conceived as an attractive extension to standard data retrieval solutions at the very early stages of fuzzy set theory and fuzzy logic development. In particular, many interesting extensions to the `SQL` language have been proposed; cf., e.g., [1]–[8]. The `SQL`'s `SELECT` instruction and its `WHERE` clause which contains a specification of the criteria that should be met by the data sought were of primary interest in these approaches. This is due to the fact that a database query may be most often identified with *conditions* that the data sought should satisfy. Such conditions may have a compound structure and comprise multiple atomic conditions imposing some constraints on the values of the attributes of the data sought. The essence of the flexible fuzzy approach to querying consists in modeling these constraints via fuzzy sets defined in the domains of the respective attributes. Another interesting feature of the proposed fuzzy extensions to regular `SQL` queries is the usage of *linguistic quantifiers* such as "most", "almost all" etc. as operators making it possible to combine conditions of a query in a more flexible, human-consistent way.

Extensions to the other clauses of the `SELECT` instruction have been also addressed even if to a slightly lesser extent. In this paper we point out the potential of such extensions to the *analytic functions* which are present in the `SQL` standard for ca. 20 years. We look at particular components of the analytic clause, a part of the `SELECT` instruction hosting an analytic function, and check how linguistic terms may make it more human consistent, i.e, more in line with reasoning and perception of a human user. We start with the standard `GROUP BY` clause of the `SELECT` instruction as it has its counterpart in the analytic clause and poses similar challenges for a flexible interpretation. Additionally, this clause has been already pointed out in the literature [9] as a promising tool in the *data summarization* process which, from our perspective, is closely related to the flexible fuzzy querying (cf., e.g., [10]). Then, we discuss particular components of the analytic clause showing some examples and proposing computational background of the proposed extended semantics of this clause. Our long term goal is to adopt this clause as a part of the authors' linguistic summaries of data [11]–[13] paradigm, in particular in the interactive, protoform based, mechanism of summaries generation.

In Section II we briefly remind the concept of a flexible fuzzy query against a relational database. Then, in Section III, we introduce a unified approach to the modeling of a fuzzy version of the `GROUP BY` clause and standard aggregate functions. Section IV is the core of this paper, discussing possible direction of the "fuzzification" of the analytic clause. We conclude with Section V. Wherever we refer to the syntax of `SQL` its Oracle™ variant is assumed.

II. The concept of a flexible fuzzy query - a brief reminder

Flexible queries are meant to provide for a more comfortable interaction of a human user with a database management system responsible for data retrieval. An important aspect of securing such a comfortable interaction is to provide a support for translating a query, which is conceived by a human user and best expressed using natural language, into a machine executable formalism such as, e.g., `SQL`. The flexible fuzzy queries provide first of all for a convenient translation of linguistic terms expressing imprecise values, relations and quantifications using the concept of a fuzzy set.

In the context of this paper it will be instructive to consider flexible fuzzy querying in the context of an interface supporting an extended version of SQL as proposed, e.g., by Kacprzyk and Zadrożny [14], [15] and Bosc et al. [16]. Basically, a traditional querying language is there extended to support *linguistic terms* in queries exemplified by fuzzy values like "young" and fuzzy relations (fuzzy comparison operators) like "much greater than" in the following SQL query (it is assumed that a *dictionary* of such linguistic terms is maintained for the user [17]):

```
SELECT *
FROM   emp
WHERE
  age IS young AND
  salary IS much greater than USD 50,000
```

The matching of a database table row against such a query is meant to be not a binary notion but has a gradual character and is identified with the membership of data to respective fuzzy sets, modeling parts of a query condition. For example, an employee represented by a row in the `emp` table matches the above query to a degree which is the minimum (operator usually assumed in fuzzy logic to model the conjunction) of the matching degrees of this employee against the conditions `age IS young` and `salary IS much greater than USD 50,000` which are, in turn, computed as the membership degree of the age of this employee to a fuzzy set representing the term `young` and the membership degree of the `salary` of this employee to a fuzzy set representing the term `much greater than USD 50,000,00`. The latter can alternatively be computed as the membership degree of a pair `(salary, 50000)` to a fuzzy relation `much greater than`.

Such a simple and straightforward extension of SQL to cover linguistic terms, represented by fuzzy sets, has been further extended in various directions, notably including *queries with linguistic quantifiers* [2], [3], [18] such as "most", "almost all", "much more than a half", etc. Linguistic quantifiers may be seen in this context as playing the role of flexible aggregation operators, acting upon satisfaction degrees of not all query conditions but, for instance, of *most* conditions.

III. AGGREGATION IN THE FUZZIFIED GROUP BY CLAUSE

The `GROUP BY` clause is an important part of the SQL `SELECT` instruction. Its basic syntax has the following form

```
  GROUP BY grouping-expression
```

and its effect is the grouping of the rows of a given table into sets of rows possessing the same value of the grouping-expression. Actually, a comma separated list of expressions may be employed instead of just one but for our purposes it will suffice to consider the above mentioned simplified version of the SQL syntax.

In the standard SQL, grouping expressions are usually of a scalar (discrete) value. For example, one may want to group hotels according to their city address or to group employees according to the department they work in. This may be equated to forming a partition of rows so as the pairs of clusters (groups) are disjoint and together cover the whole set of rows. Thus, a motivation similar to the one laying ground for fuzzy clustering may be adopted to justify the need for the fuzzification of the `GROUP BY` clause. Namely, one may need to group the rows but it is possible and natural for some rows to belong simultaneously to different groups, usually to a varying degree. For example, one may group hotel rooms according to their price matching with linguistic terms such as, e.g., "cheap", "middle-priced", "expensive". Then, a room may belong to cheap ones to some degree but at the same time possibly it belongs also to the group of the medium priced rooms to another degree. Similar examples for the employees may group them according to their salary matching various, linguistically expressed, levels (e.g., "low", "average", "high", "very high") or according to their age ("young", "middle-aged", "old"). These examples provide for another justification to "fuzzify" the `GROUP BY` clause. Namely, the grouping of rows is often especially attractive with respect to the continuous attributes (columns) such as `price`, `salary`, `age`. However, most often it would be impractical to directly group rows in the standard way according to such attributes values as the resulting groups would be often small in size or even singletons. This of course depends on the actual distribution of given column values in the table. For example, if there is a rule that there are only, e.g., 5 distinct levels of the salaries in a company then direct grouping with respect to this attribute may make sense. Anyway, grouping rows with respect to fuzzy (linguistic) terms seems to be a solution in a more general case of continuous attributes.

The `GROUP BY` clause is attractive as it provides for a different perspective on data gathered in a given table what obtains via the grouping itself and via applying *aggregate functions* to particular groups of rows. Let us consider a following example of the "crisp" SQL query involving the `GROUP BY` clause:

```
SELECT depno, COUNT(*)
FROM emp                          (1)
GROUP BY depno
```

This query, even if it gets data from the `emp` table containing data on the *employees* of a company, actually provides the user with information on departments which these employees work in. It is worth noticing that this information on departments is not, and should not be, represented directly in the `dept` table which usually accompanies the `emp` table in a company's database and contains basic information on departments such as the name, contact info etc.

A query shown in (1) tells how many employees work in each department, thanks to the use of the aggregate function `COUNT` in the `SELECT` clause. As mentioned earlier it may be very useful to apply in such a query some fuzzy (linguistically expressed) criteria of grouping rows. For example, one may be interested in:

What is the count or average salary in, e.g., three age groups comprising young, middle-aged and old employees. (2)

Or, what is the number of 5 star hotels in small, medium size and large cities in Scotland, and what is an average price of the room in those hotels in each mentioned category of cities.

The "fuzzifying" of the `GROUP BY` clause was however treated at the beginning as more challenging than, e.g., of the `WHERE` clause. In fact neither SQLf [1], [19] nor FQUERY for Access [17], [20], [21] originally provided for including fuzzy terms into this clause. Bosc et al. [1] considered only fuzzy conditions in the related `HAVING` clause which may be used to filter out some groups of rows produced by the `GROUP BY` clause. However, originally they assumed the grouping in its crisp version and only later on [9] introduced grouping rows with respect to a fuzzy partition; cf. also [22]–[25].

In order to comprehensively "fuzzify" the `GROUP BY` clause one has to address two main problems:

1) how to define "fuzzy" grouping syntactically and semantically,
2) how to execute aggregate operators against resulting "fuzzy groups" of rows. (3)

The first challenge is of a more technical nature. As soon as the `WHERE` clause allows for fuzzy conditions then producing fuzzy groups, e.g., corresponding to (2), is conceptually simple as the required groups of rows may be produced by using three separate `SELECT` instructions with the `WHERE` clauses featuring the conditions "age IS young", "age IS middle" and "age IS old", respectively. So the challenge is limited to appropriately extending the `GROUP BY` clause so as to properly accommodate the interpreted grouping-expression. Of course, one also has to devise a scheme of an efficient execution of a query containing such a fuzzy `GROUP BY` clause.

The second problem mentioned above is of a more theoretical nature. Its essence is best illustrated using our example (2). Namely, how should one understand the concept of an average salary of a group of employees when each of them can belong to this group to some degree which is based on its age matching with a given group label (linguistic term), i.e., with being young, middle-aged or old ? How to define and compute such an aggregate value and how to make an approach general enough so as it covers also other aggregate operators known in `SQL` such as `MAX`, `MIN`, etc.

Let us now address both problems mentioned in (3) in a slightly more detailed way.

Let us assume that the `GROUP BY` clause takes the following form:

```
GROUP BY price
```
(4)

and consider how a fuzzy interpretation may be adopted for it.

First of all, it should be decided where linguistic terms, such as " cheap" or "expensive", to be used to group rows, do come from. In view of a brief reminder of flexible fuzzy querying provided in Section II, a natural source of such terms is a dictionary of linguistic terms maintained by a fuzzy querying interface; cf., e.g., [17], [26]. Effectively, it boils down to the definition of a given attribute (column) as a linguistic variable [27]. Alternatively, such a fuzzy partition of a given attribute domain (here: `price`) may be automatically derived in querying environment using, e.g., a fuzzy c-means algorithm; cf. also [28]. In [9] the authors propose to extend the `GROUP BY` clause so as it can include an explicit specification of the (fuzzy) partition of an attribute domain with respect to which the grouping takes place.

No matter how the linguistic terms set, $lts_A = \{l_1^A, \ldots, l_k^A\}$, is established for a given attribute A, for each row t the matching degree of A's value, $t(A)$, with a particular linguistic term $l_i^A \in lts_A$ is denoted as $md(t(A), l_i^A)$ and computed as the membership degree of $t(A)$ to the fuzzy set representing the particular linguistic term:

$$md(t(A), l_i^A) = \mu_{L_i^A}(t(A)) \tag{5}$$

where L_i^A denotes a fuzzy set representing the linguistic term l_i^A and $\mu_{L_i^A}$ is its membership function. Then, a group of rows G_i is formed for every linguistic term l_i^A and it is a fuzzy set of rows with the following membership function:

$$\mu_{G_i}(t) = md(t(A), l_i^A) \tag{6}$$

Some groups G_i may be empty (non-existent) and some pairs of groups may have non-empty intersection. Additionally, each row t is assumed to be equipped with an additional attribute (column) expressing its membership to a group G_i, equal $md(t(A), l_i^A)$. This line of reasoning provides a solution to the first problem of (3).

In order to address the second problem of (3) and make the "fuzzy" `GROUP BY` clause operational one has to define how particular aggregate functions of the standard `SQL` are going to work for fuzzy groups of rows. As it was already mentioned, in standard `SQL` one uses grouping of rows in order to compute some aggregate values for the obtained groups of rows. For example, the user may be interested in learning *how many* hotels there are in particular cities, what *average* salary employees earn in particular departments, or what the *maximum* salary in particular departments is. Then, one uses an `SQL` query of the following form:

```
SELECT grouping-expression,
       AGG(aggregated-expression)
FROM table
GROUP BY grouping-expression
```
(7)

where `AGG` is a placeholder for a specific aggregate function, such as `COUNT`, `SUM` or `AVG` . Again, a query scheme shown in (7) is a simplification of the general `SQL` syntax but is detailed enough to discuss the question of its fuzzy interpretation. For some aggregate functions `AGG` this interpretation is fairly obvious and easy, for some others it is more difficult and not that obvious. We will first briefly discuss in the following subsections two classes of aggregate functions, represented by COUNT and AVG, respectively; cf. also [26].

It is worth noticing that even for a "non-fuzzy" GROUP BY clause, i.e. based on a crisp grouping expression, the aggregate functions still have to be prepared to deal with fuzzy (multi)sets of values. Namely, if the WHERE clause contains fuzzy conditions and a crisp GROUP BY clause is used then the rows in the "crisp" groups are assigned matching degrees with respect to the mentioned WHERE clause conditions and thus aggregating applies, in general, to fuzzy multisets.

Let us notice here that a presumably simplest solution consists in ignoring the fuzzy character of the set of aggregated values and applying aggregation functions to the support of a given fuzzy multiset. This approach has been adopted by Bosc and Pivert [1] in their earliest version of the SQLf language where GROUP BY clause was not to allowed to be fuzzified but aggregate functions in the SELECT clause were allowed to be used together with fuzzy conditions in the WHERE clause.

There is a vast literature on fuzzy aggregation operators meant as counterparts of standard logical connectives. The problem of "fuzzy data" aggregation in the framework of data querying has been also dealt with, cf., e.g., the works of Bosc and his group [22]–[25], the paper of Laurent [29] discussing it in a broader context as well as works of Dubois and Prade focused on theoretical fundations [30], [31].

A. *COUNT aggregate function*

This aggregate function forms a class of itself and, in the original, non-fuzzy, context it just returns the number of rows in a given group. We will consider here only its COUNT(*) form as from the point of view of a fuzzy interpretation it does not differ from the other versions. Its fuzzy interpretation is thus quite straightforward - it is assumed that it returns the fuzzy cardinality of the given group while the matching degree defined in (6) is interpreted as the membership degree of particular rows to a given group. Formally, assuming the cardinality proposed by Zadeh, i.e., the so-called ΣCount [32], one obtains for a group G_i, corresponding to a linguistic term l_i defined over attribute's A domain, the following result:

$$\texttt{COUNT(*)} \longmapsto \sum_{t_j} \mu_{L_i^A}(t_j(A)) \tag{8}$$

Thus, in case of the COUNT aggregate function its fuzzy interpretation is quite clear even if several other definitions of fuzzy set cardinality may be employed; cf., e.g., [33].

B. *AVG aggregate function*

This class of aggregate functions is more challenging when one tries to apply it to a fuzzy set. In order to study possible approaches let us consider that some numbers forming a fuzzy multiset, $\widetilde{X}$, have to be averaged. A multiset has to be considered as the values being aggregated may be repeated.

A simple approach consists in some generalization of the standard averaging formula. Namely, for a crisp multiset of numbers, $X = \{x_i\}$, their average is computed as follows:

$$\texttt{AVG}\ (X) = \frac{\sum_{x_i \in X} x_i}{|X|} \tag{9}$$

where $|X|$ denotes the cardinality of a crisp set X.

Then, in case a fuzzy set $\widetilde{X}$ replaces X in (9) it is natural to use a fuzzy set cardinality, in particular the ΣCount in the denominator of the formula (9). It remains unclear how to deal with the sum in the numerator of (9). A reasonable approach might be to replace the sum with a weighted sum where the membership degrees play the role of the weights and thus one comes up with the following formula [26]:

$$\texttt{AVG}\ (\widetilde{X}) = \frac{\sum_j \mu_{\widetilde{X}}(x_j) * x_j}{\sum_j \mu_{\widetilde{X}}(x_j)} \tag{10}$$

what may be further rewritten as:

$$\texttt{AVG}\ (\widetilde{X}) = \sum_j p_j * x_j \tag{11}$$

where p_j denotes the normalized membership degree of x_j (or, more precisely, the normalized membership degree of the row of which x_j is a value of an attribute under consideration), i.e.:

$$p_j = \frac{\mu_{\widetilde{X}}(x_j)}{\Sigma_k \mu_{\widetilde{X}}(x_k)} \tag{12}$$

C. *WOWA operator based approach*

However, in what follows we adopt a more general approach to aggregate functions which consists in using Torra's Weighted Ordered Weighted Averaging Operators (WOWA) [34]. WOWA is an extension to Yager's Ordered Weighted Averaging operator (OWA) [35], [36] and is defined as follows [34], for parameters: n being a dimension of the operator, and $\mathbf{w} = [w_1, \ldots, w_n]$, $\mathbf{p} = [p_1, \ldots, p_n]$ being vectors of weights, and for a vector of arguments $\mathbf{a} = [a_1, \ldots, a_n]$ to be aggregated:

$$f^{\mathbf{w},\mathbf{p}}_{WOWA}(a_1, \ldots, a_n) = \sum_i \omega_i * a_{\sigma(i)} \tag{13}$$

where σ is a permutation of $\{1, \ldots, n\}$ such that $\forall i \in \{1, \ldots, n-1\}$ $a_{\sigma_{i+1}} \leq a_{\sigma_i}$ and the *weights* ω_i are defined as:

$$\omega_i = w^\star(\Sigma_{j \leq i}\ p_{\sigma(j)}) - w^\star(\Sigma_{j<i}\ p_{\sigma(j)}) \tag{14}$$

where $w^\star$ is a monotonic increasing function interpolating the point $(0, 0)$ together with the points $(i/n, \Sigma_{j \leq i} w_j)$ $\forall i \in \{1, \ldots, n\}$ ($w^\star$ is required to be a straight line when these points lie along such a line).

Thus the WOWA operator makes it possible to use two types of weights. The weights of the original OWA operator, denoted w_i:

$$\mathbf{w} = [w_1 \ldots, w_n], \quad \forall_{i \in \{1,\ldots,n\}}\ w_i \in [0, 1] \text{ and } \sum_i w_i = 1 \tag{15}$$

which are assigned to the positions of the arguments in their non-increasing ordering, i.e., w_1 is the weight of the largest among the arguments $\{a_i\}_{i \in \{1,\ldots,n\}}$, w_2 of the second largest etc. Depending on the choice of these weights one obtains various aggregation operators covering the whole spectrum from the minimum operator via arithmetic average, up to the maximum operator. The weights of the second type, denoted p_i:

$$\mathbf{p} = [p_1 \dots, p_n], \quad \forall_{i \in \{1,\dots,n\}} \; p_i \in [0,1] \text{ and } \sum_i p_i = 1 \tag{16}$$

are assigned to particular arguments being aggregated, a_i, and not to their position in any ordering.

Thus, in the setting considered in this paper, the weights w_i make it possible to define a given aggregate function (e.g., $\mathbf{w} = [1, 0, \dots, 0]$ for the maximum operator, or $\mathbf{w} = [1/n, 1/n, \dots, 1/n]$ for the arithmetic average operator) while the weights p_i make it possible to take into account the membership degrees of the rows forming a group. In order to satisfy (16) the normalization shown in (12) is adopted. It is worth noticing that although an WOWA operator is parametrized by two vectors of weights those weights have a straightforward interpretation when considered for the purposes of modelling SQL aggregation operator applied to a fuzzy set of rows.

It may be easily shown that the WOWA operator emulating the arithmetic average operator, i.e., with the weights $\mathbf{w} = [1/n, \dots, 1/n]$ and $\mathbf{p}$ computed as in (12) yields exactly the same results as the operator (11) proposed in [26].

Thus, in general the WOWA operator makes it possible to apply any aggregation operator definable with the use of an OWA operator to a fuzzy set of arguments and one obtains a single number as the result. Moreover, the WOWA operator has some desired properties, For any weighting vectors $\mathbf{w}$ and $\mathbf{p}$, satisfying (15) and (16), the WOWA operator is an *aggregation operator*, i.e., yields results between the minimum and maximum operators [34]. For example, when $\forall i \; p_i = 1/n$, where n is the dimension of the operator, then it acts as the OWA operator with the weight vector $\mathbf{w}$. Thus, if a group of rows is non-fuzzy then the aggregate function modeled by the given WOWA operator behaves as expected - as the original aggregate operator. However, due to the normalization (12) the WOWA based aggregate operator produces the same results for many different fuzzy sets $\tilde{X}$ (cf. (12)), provided that the vectors $\mathbf{a}$ and $\mathbf{w}$ are fixed (the set $\tilde{X}$ is a fuzzy set of arguments a_i with the membership degrees p_i). This may be less intuitive but this is, in general, the case of any possible interpretations of the results of a fuzzy set aggregation. For example, the WOWA operator of dimension n representing the maximum operator (i.e., with $\mathbf{w} = [1, 0, \dots]$) when applied to a vector of arguments $\mathbf{a}$ using a vector of weights $\mathbf{p}$ such that not all p_i's are equal $1/n$ may produce as a result a value which does not appear in $\mathbf{a}$.

Among the standard aggregate functions in `SQL` there is also `SUM` which produces the sum of the expression computed over a group of rows. It is not an aggregation operator in the sense mentioned earlier and thus does not directly fit into the aggregation scheme provided by the WOWA operator. However one may interpret it in the same way as the aggregation operators exploiting an obvious relation between `SUM` and `AVG` aggregate functions which is:

$$SUM(exp) = n * AVG(exp)$$

where n is the number of rows in a group (i.e., a fuzzy set $\tilde{X}$) over which the expression `exp` is evaluated.

Based on that let us define the `SUM` operator over a fuzzy set (a group) of n rows in the following way:

$$SUM(exp) = (\sum_i \mu_{\tilde{X}}(x_i)) * f^{\mathbf{w},\mathbf{p}}_{WOWA}(x_1, \dots, x_n) \tag{17}$$

where x_i is the value of expression exp for i-th element of the group (represented by a fuzzy set $\tilde{X}$), $\mathbf{w} = [1/n, \dots, 1/n]$ and $\mathbf{p} = [\mu_{\tilde{X}}(x_1)/(\sum_i \mu_{\tilde{X}}(x_i)), \dots, \mu_{\tilde{X}}(x_n)/(\sum_i \mu_{\tilde{X}}(x_i))]$, as previously. This way some basic expected properties of the `SUM` aggregate function are preserved. Namely, if it is computed for a group of rows represented by a crisp set ($\mu_{\tilde{X}}(x_i) = 1 \; \forall i$) then the result is a standard sum. From the computational point of view, formula (17) is equivalent to a simple weighted sum:

$$SUM(exp) = \sum_i \mu_{\tilde{X}}(x_i) * x_i \tag{18}$$

and, of course, this form should be used in an implementation.

IV. Analytic functions

A. Basics

Analytic functions in `SQL` are, in fact, aggregate functions but used in a different context. Namely, aggregate functions are computed for a group of rows, defined by a `GROUP BY` clause, or for all rows in a table. Thus, they produce an aggregated value for a set of rows. Analytic functions go beyond that, extending the repertoire of the aggregation schemes and, most importantly, making it possible to compute the aggregated value in the context of a single row. Hence, the set of of values (rows) to be aggregated, referred to as a *window*, is determined for each row separately. A call to an analytic function is thus a part of what is called an *analytic clause*.

Let us start with briefly reminding the syntax of an analytic clause in SQL (it is again a simplified version of the syntax but it will serve our purposes):

```
AGG(aggregated-expression) OVER
     (
       PARTITION BY grouping-expression
       ORDER BY ordering-expression
       <windowing-clause>
     )
)
```

where `AGG` denotes an analytic function, the grouping expression defines the grouping of the table rows as in the case of the `GROUP BY` clause, ordering expression determines how rows are ordered within a given group and the windowing clause decides which part of the group will be subject to the aggregation carried out by the analytic function.

Thus, the analytic clause comprises the following parts, all of which are optional:

1) a partition of the rows into groups (`PARTITION BY` subclause); it works like the `GROUP BY` clause but partitioning is executed for each row t separately and

only a group G_t to which t belongs is considered while computing the analytic function; if the partition is not specified then the whole table plays the role of G_t;
2) an ordering of the rows in the window which is important for some analytic functions as well as helps to define the window mentioned below (`ORDER BY` subclause); if the ordering is used alone, i.e., without a window (see below), then it yields a default window which comprises only the rows in the group with the same or lower value of the ordering expression than the current row has,
3) a window of rows (`<windowing-clause>`); a subset of a group G_t comprising rows which participate in computing the analytic function value; window size can be defined either in terms of a number of rows or in terms of the values of the ordering expression.

Now let us look at possible interpretations of linguistic terms within the components of the analytic clause.

B. Partitioning

As mentioned earlier, this part of the analytic clause resembles, both with respect to the syntax and the semantics, the `GROUP BY` clause of the `SELECT` instruction. Thus its fuzzification may be carried out along the lines discussed in section (III), in particular by using a linguistic term in the role of a grouping expression. However, there is an important difference between the `GROUP BY` and the `PARTITION BY` clauses when a fuzzy grouping expression is used. In the case of the former a row may be assigned to a few (fuzzy) groups to different degrees. For each resulting fuzzy group the value of an aggregate function is computed. On the other hand, in case of the `PARTITION BY` clause for a given row t an analytic function is computed in the context of one group, G_t, to which row t “belongs”. In the fuzzy case this belongingness is, in general, not well defined as t may belong to several groups to a different degree.

We propose two solutions:

1) a group is chosen for which t’s membership is highest; if there is more than one such a group then these groups are combined,
2) an analytic function is computed in the context of each group, using an appropriate WOWA operator, and then computed values are aggregated using an appropriately adjusted WOWA operator.

The former solution does not need any further explanations. For the latter, let us consider the following example:

```
SELECT
  name, salary,
  AVG(salary) OVER (
            PARTITION BY ~age) AS agvsal
FROM emp
```

which is meant to retrieve for each employee his or her name, salary and an average salary for his or her age group. Here and in what follows we assume that the ˜ symbol preceding a column name triggers a fuzzy grouping based on the linguistic terms defined for a given column (here, as previously: `young`, `middle-aged`, `old`).

Let John Doe belong to the groups of middle-aged and young employees to the degrees, respectively, 1 and 0.7. Then, for him the third expression (`avgsal`) in the `SELECT` clause above will be computed as follows. First, the WOWA operators will aggregate salaries in groups (fuzzy sets) gathering, respectively, `young` and `middle-aged` employes using weights $\mathbf{w}$ and $\mathbf{p}$ properly defined to take into account the number of employees with non-zero membership degrees in each group (implying the dimension of the WOWA operator), and their matching degrees with the linguistic terms `young` and `middle-aged` (defining the $\mathbf{p}$ vector). Let us assume that these WOWA operators produced, respectively, the values b_{young} and $b_{\text{middle-aged}}$. Then, the final value of the `avgsal` for John Doe will be computed using the following expression:

$$f_{WOWA}^{\mathbf{w},\mathbf{p}}(b_{\text{young}}, b_{\text{middle-aged}}) \tag{19}$$

where:

$$\mathbf{w} = [1/2, 1/2]$$

$$\mathbf{p} = [\mu_{G_{\text{young}}}(t_{\text{John Doe}}), \mu_{G_{\text{middle-aged}}}(t_{\text{John Doe}})]$$

G_{l_i} denote (fuzzy) groups of rows resulting from the `PARTITION BY  age` clause, and $t_{\text{John Doe}}$ denotes a row representing John Doe. Thus, the weighted average of b_{young} and $b_{\text{middle-aged}}$ with weights corresponding to the degrees of how John Doe is young and how middle-aged, respectively.

C. Ordering

The ordering of rows based on a linguistic term, contrary to the use of linguistic terms in the `GROUP BY` or `PARTITION BY` clause, does not seem reasonable. For example, the ordering of employees according to their age in such a way that their actual age is replaced with linguistic terms such as “young”, “middle-aged”, “old” etc., and only then the rows are sorted, may only make some rows (employess) indistinguishable with respect to the value of an ordering expressions what rarely will be desired. On the other hand, it should be noted that the ordering applies, in general, to a fuzzy set of rows as a fuzzy condition may be used in the `WHERE` clause of the `SELECT` instruction and the ordering of the rows takes place at the end of the query processing. Thus, it certainly may be of interest to order the rows according to the membership degree which, in fact, should be treated as an additional attribute and it should be possible to use it in ordering expression. On the other hand, it is worth a study if in some scenarios the membership degrees of the rows should be taken into account when the rows are ordered with respect to another regular ordering expression. For example in case of the following query, when the user wants to rank middle-aged employees according to their seniority:

```
SELECT name, age, seniority
FROM emp
WHERE age IS middle-aged
```

```
ORDER BY senority
```

it might be reasonable to take into account how particular employees match a linguistic term “middle-aged” while ordering them. This is an example of a query with ordering concerning the whole table but the same applies to a group of rows yielded by the `PARTITION BY` part of the analytic clause.

An interesting option may be ordering the rows according to their matching degree against a selected linguistic term (a fuzzy value). For example, ordering employees according to the degree their age matches the linguistic term “middle-aged” may be of interest as the membership function of a fuzzy set representing this linguistic term usually will not be monotonic with respect to the value of an underlying attribute, i.e., the age.

D. Definition of a window

The windowing clause may appear in the analytic clause only together with the ordering clause as a window may be defined only with respect to some ordering of rows in a group. The most interesting variants of the windowing clause syntax are from our perspective the following:

```
ROWS BETWEEN n PRECEDING/FOLLOWING AND
             m PRECEDING/FOLLOWING

RANGE BETWEEN x PRECEDING/FOLLOWING AND
              y PRECEDING/FOLLOWING
```

Both versions of the syntax define the window with respect to a row t, referred to as the *current row*, which is the row for which an analytic function is computed. The first version makes it possible to define a window as a set of subsequent rows located between n-th position before/after the current row and m-th position before/after the current row. The second version indicates as the first row of the window the row for which the value of the ordering expression is lower/higher by x compared to the value of this expression at the current row, and similarly for the last row of the window.

Both variants of the windowing clause syntax are readily available for the “fuzzification”. Let us consider some examples. The following query shows the average price paid for each hotel room by its *latest* guests:

```
SELECT
  roomnumber,
  AVG(price} OVER(
     PARTITION BY roomnumber
     ORDER BY checkindate
     ROWS BETWEEN ˜latest PRECEDING
             AND
                  1 PRECEDING) avgprice
FROM roomsoccupancy
```

where “latest” is a fuzzy value represented by, e.g., the following membership function:

$$\mu_{latest}(x) = \begin{cases} 1 & \text{for } x \le 2 \\ 0.83 & \text{for } x = 3 \\ 0.33 & \text{for } x = 4 \\ 0.17 & \text{for } x = 5 \\ 0.0 & \text{for } x > 5 \end{cases} \tag{20}$$

i.e., up to two stays before the given current stay are treated as fully “latest”, third stay before the current is “latest” to the degree 0.83 etc.

The following query shows for each employee how much lower is his or her salary compared to the best earning employee at *more or less* the same age:

```
SELECT
  name,salary,birthdate,
  MAX(salary) OVER(
    ORDER BY birthdate ASC
    RANGE BETWEEN
       Not_much_more_than_730 PRECEDING AND
       Not_much_more_than_730 FOLLOWING) -
  salary
FROM emp
```

where “Not_much_more_than_730” is a fuzzy value (linguistic term) represented by a fuzzy set A with, e.g., the following membership function:

$$\mu_A(x) = \begin{cases} 1 & \text{for} \quad x \le 800 \\ (1000 - x)/200 & \text{for} \quad 800 < x \le 1000 \\ 0 & \text{for} \quad x > 1000 \end{cases} \tag{21}$$

which represents the meaning of the statement that given number of days is not much greater than 730.

The computation of values of the analytic functions against a window defined in above mentioned flexible way follows the scheme presented for the `GROUP BY` and `PARTITION BY` clauses (cf. section III and subsection IV-B). The weights $\mathbf{p}$ are now implied by the membership degree of a row to a respective window. In the most complex scenario these weights may be the conjunction of weights implied by the `WHERE` clause, the `PARTITION BY` clause and the window.

Both queries show that using windows with flexible limits is of practical value. One may expect that this is even more useful when such windows are part of a linguistic summary.

V. Concluding remarks

We have proposed a novel concept of a flexible analytic clause of the `SQL`’s `SELECT` instructions to make them more flexible and human consistent. First of all, it provides for a considerable extension of applicability of the new flexible fuzzy queries themselves, by making them easier to use and better representing human intentions and preferences. Moreover, the use of the new flexible analytic clause should make it possible to considerably extend the classes of linguistic data summaries along the ideas of using for this purpose relations between flexible fuzzy queries and linguistic data summaries

[10] or relations between bipolar queries and bipolar (contextual) linguistic data summaries [37]. This should provide for an additional functionality of the powerful concept of a linguistic data summary.

This paper is a first step towards defining a flexible fuzzy interpretation of the analytic clause. Some details of the syntax and semantics have to be further elaborated as well as an effective scheme of execution has to be studied. In particular, desired properties of the emerging operators have to be dealt with in a more comprehensive way and their satisfying by proposed alternative approaches studied in a more depth. This includes also a study of an appropriate interpretation of many different analytic functions, beyond `AVG` , `MAX` etc. The concept sketched in this paper looks promising and has a potential in extending the applicability of the linguistic summarization of data.

References

[1] P. Bosc, M. Galibourg, and G. Hamon, "Fuzzy querying with SQL: extensions and implementation aspects," *Fuzzy Sets and Systems*, vol. 28, pp. 333–349, 1988.

[2] J. Kacprzyk and A. Ziółkowski, "Database queries with fuzzy linguistic quantifiers," *IEEE Transactions on System, Man and Cybernetics*, vol. 16, no. 3, pp. 474–479, 1986.

[3] J. Kacprzyk, S. Zadrożny, and A. Ziółkowski, "FQUERY III+: a "human consistent" database querying system based on fuzzy logic with linguistic quantifiers," *Information Systems*, vol. 14, no. 6, pp. 443–453, 1989.

[4] J. Galindo, Ed., *Handbook of Research on Fuzzy Information Processing in Databases*. IGI Global, 2008.

[5] G. De Tré and S. Zadrożny, "Soft computing in database and information management," in *Springer Handbook of Computational Intelligence*, J. Kacprzyk and W. Pedrycz, Eds. Springer, 2015, pp. 295–312.

[6] P. Bosc and J. Kacprzyk, Eds., *Fuzziness in Database Management Systems*. Heidelberg: Physica-Verlag, 1995.

[7] O. Pivert and P. Bosc, *Fuzzy Preference Queries to Relational Database*. Imperial College Press, 2012.

[8] O. Pivert and S. Zadrożny, Eds., *Flexible Approaches in Data, Information and Knowledge Management*, ser. Studies in Computational Intelligence. Springer, 2014, vol. 497.

[9] P. Bosc, O. Pivert, and G. Smits, "On a fuzzy group-by and its use for fuzzy association rule mining," in *Advances in Databases and Information Systems*, B. Catania, M. Ivanović, and B. Thalheim, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 88–102.

[10] J. Kacprzyk and S. Zadrożny, "Data mining via linguistic summaries of data: an interactive approach," in *Methodologies for the Conception, Design and Application of Soft Computing. Proceedings of IIZUKA'98*, Yamakawa T. and Matsumoto G., Eds., Iizuka, Japan, 1998, pp. 668 – 671.

[11] J. Kacprzyk and R. R. Yager, "Linguistic summaries of data using fuzzy logic," *International Journal of General Systems*, no. 30, pp. 33–154, 2001.

[12] J. Kacprzyk and S. Zadrozny, "Fuzzy logic-based linguistic summaries of time series: a powerful tool for discovering knowledge on time varying processes and systems under imprecision," *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, vol. 6, no. 1, pp. 37–46, 2016. [Online]. Available: https://doi.org/10.1002/widm.1175

[13] ——, "Computing with words is an implementable paradigm: Fuzzy queries, linguistic data summaries, and natural-language generation," *IEEE Trans. Fuzzy Systems*, vol. 18, no. 3, pp. 461–472, 2010. [Online]. Available: https://doi.org/10.1109/TFUZZ.2010.2040480

[14] J. Kacprzyk and S. Zadrożny, "The paradigm of computing with words in intelligent database querying," in *Computing with Words in Information/Intelligent Systems. Part 1. Foundations. Part 2. Applications*, L. Zadeh and J. Kacprzyk, Eds. Heidelberg and New York: Springer-Verlag, 1999, pp. 382–398.

[15] ——, "Computing with words in intelligent database querying: standalone and internet-based applications," *Information Sciences*, vol. 134, no. 1-4, pp. 71–109, 2001.

[16] P. Bosc and O. Pivert, "SQLf: A relational database language for fuzzy querying," *IEEE Transactions on Fuzzy Systems*, vol. 3, no. 1, pp. 1–17, 1995.

[17] J. Kacprzyk and S. Zadrożny, "FQUERY for Access: fuzzy querying for a windows-based DBMS," in *Fuzziness in Database Management Systems*, P. Bosc and J. Kacprzyk, Eds. Heidelberg: Physica-Verlag, 1995, pp. 415–433.

[18] ——, "Linguistic database summaries and their protoforms: towards natural language based knowledge discovery tools," *Information Sciences*, vol. 173, no. 4, pp. 281–304, 2005.

[19] P. Bosc and O. Pivert, "SQLf: a relational database language for fuzzy querying," *IEEE Trans. Fuzzy Systems*, vol. 3, no. 1, pp. 1–17, 1995. [Online]. Available: https://doi.org/10.1109/91.366566

[20] S. Zadrozny and J. Kacprzyk, "FQUERY for access: towards human consistent querying user interface," in *Proceedings of the 1996 ACM Symposium on Applied Computing, SAC'96, Philadelphia, PA, USA, February 17-19, 1996*, K. M. George, J. H. Carroll, D. Oppenheim, and J. Hightower, Eds. ACM, 1996, pp. 532–536. [Online]. Available: https://doi.org/10.1145/331119.331446

[21] J. Kacprzyk and S. Zadrożny, "SQLf and fquery for access," in *Proceedings of the Conference IFSA/NAFIPS 2001*, Vancouver, Canada, 2001, pp. 2464–2469.

[22] P. Bosc, O. Pivert, and L. Lietard, "Aggregate operators in database flexible querying." in *Proceedings of the IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2001)*, Melbourne, Australia, 2001, pp. 1231–1234.

[23] P. Bosc, L. Lietard, and O. Pivert, "Sugeno fuzzy integral as a basis for the interpretation of flexible queries involving monotonic aggregates." *Information Processing & Management*, vol. 39, no. 2, pp. 287–306, 2003.

[24] P. Bosc, O. Pivert, and L. Litard, "On the comparison of aggregates over fuzzy sets," in *Intelligent Systems for Information Processing*, B. Bouchon-Meunier, L. Foulloy, and R. R. Yager, Eds. Amsterdam: Elsevier Science, 2003, pp. 141 – 152.

[25] P. Bosc and L. Lietard, "Aggregates computed over fuzzy sets and their integration into sqlf," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 16, no. 06, pp. 761–792, 2008.

[26] G. Nowakowski, "Fuzzy queries on relational databases," in *2018 International Interdisciplinary PhD Workshop (IIPhDW)*, May 2018, pp. 293–299.

[27] L. Zadeh, "The concept of a linguistic variable and its application to approximate reasoning. Part I-III." *Information Sciences*, vol. 8,8,9, pp. 199249,301357,4380, 1975.

[28] C. Li, M. Wang, L. Lim, H. Wang, and K. C. Chang, "Supporting ranking and clustering as generalized order-by and group-by," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*, C. Y. Chan, B. C. Ooi, and A. Zhou, Eds. ACM, 2007, pp. 127–138. [Online]. Available: https://doi.org/10.1145/1247480.1247496

[29] A. Laurent, "Fuzzy multidimensional databases," in *Uncertainty Approaches for Spatial Data Modeling and Processing: A Decision Support Perspective*, ser. Studies in Computational Intelligence, J. Kacprzyk, F. E. Petry, and A. Yazici, Eds. Springer, 2010, vol. 271, pp. 43–59.

[30] D. Dubois and H. Prade, "Measuring properties of fuzzy sets: A general technique and its use in fuzzy query evaluation," *Fuzzy Sets and Systems*, vol. 38, no. 2, pp. 137 – 152, 1990.

[31] ——, "Scalar evaluations of fuzzy sets: overview and applications," *Applied Mathematics Letters*, vol. 3, no. 2, pp. 37–42, 1990.

[32] L. Zadeh, "A computational approach to fuzzy quantifiers in natural languages," *Computers and Mathematics with Applications*, vol. 9, pp. 149–184, 1983.

[33] M. Wygralak, *Cardinalities of Fuzzy Sets*, ser. Studies in Fuzziness and Soft Computing. Springer, 2003, vol. 118.

[34] V. Torra, "The weighted OWA operator," *International Journal of Intelligent Systems*, vol. 12, pp. 153–166, 1997.

[35] R. Yager, "On ordered weighted averaging aggregation operators in multi-criteria decision making," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 18, pp. 183–190, 1988.

[36] R. Yager and J. Kacprzyk, Eds., *The Ordered Weighted Averaging Operators: Theory and Applications*. Boston: Kluwer, 1997.

[37] M. Dziedzic, S. Zadrożny, and J. Kacprzyk, "Towards bipolar linguistic summaries: a novel fuzzy bipolar querying based approach," in *FUZZ-IEEE 2012, IEEE International Conference on Fuzzy Systems, Brisbane, Australia, June 10-15, 2012, Proceedings.* IEEE, 2012, pp. 1–8.