This is the accepted version of the article with DOI 10.1109/FUZZ62266.2025.11152119

Fuzzy Group By Queries via RL-Instances in Conventional RDBMS

Nicolás Marín[a,c], Gustavo Rivas-Gervilla[b], and Daniel Sánchez[a,c]

[a]Department of Computer Science and Artificial Intelligence, University of Granada, Spain
[b]Department of Statistics and Operations Research, University of Granada, Spain
[c]CITIC-UGR, University of Granada, Spain

Abstract: SQL *group by* queries allow grouping tuples in relational databases and computing aggregated values from the obtained sets of tuples, which makes them a particularly useful tool for data analysis. In this paper, we propose a way to solve this type of queries in a fuzzy environment, using fuzzy partitions of the domain of attributes according to Ruspini. The resolution of the query is carried out through the use of RL-instances and concepts from the theory of Representations by Levels, particularly a new method to derive RL-partitions from fuzzy partitions. The approach can be easily used in conventional systems through well-known SQL patterns.

1 Introduction

Data are the fundamental raw material in many processes that occur in companies. These data are often structured as collections of interconnected relations under Codd's Relational data model, a data model whose robustness, derived from the first-order predicate logic, has made it the data model par excellence in databases. Relational database management systems (RDBMS) allow us to create this type of stores and to efficiently interact with them through languages such as SQL, a standard language that any information systems professional knows and handles.

In the area of Flexible Querying, significant efforts have been made to develop advanced SQL-based tools in order to make it easier for users to ask queries about stored data, and to offer them understandable answers according to their preferences. Within this context, tools that allow elements of natural language to be incorporated into queries stand out, many of them developed based on results derived from Zadeh's Fuzzy Set Theory.

Thanks to this large research work, there is a wide variety of proposals in the literature that allow working reliably with fuzziness in relational databases (see, for example, [Gal08; KZD15; PS21; PZ14; TZ15]). However, all of these proposals either require the development from scratch of a system that supports the proposed extended version of the SQL language, or they require the installation of complex libraries to incorporate the new query capabilities. Additionally, developers and/or users have to face the choice between a wide variety of operators for solving the queries affected by graduality, with known limitations in some cases to preserve certain properties derived from Boolean algebras [CMS22; DP80].

Proposals such as those available in [Now18; Now19] permit the resolution of fuzzy queries with standard SQL, avoiding the development of new systems and languages. In the same direction and with the additional objective of allowing the use of conventional SQL patterns in the resolution of fuzzy queries, a new proposal has recently appeared that follow a different approach to handling fuzzy queries on relational databases [CMS22]. This approach is based on the use of RL-instances as a complement to the fuzzy instances[1] that underpin a good part of the aforementioned fuzzy querying proposals. RL-instances use the ideas of the Representation by Levels theory [S+12] to manage fuzzy queries, with important advantages: they can be used very simply in a conventional relational database management system; they allow us to resolve fuzzy queries using standard SQL query patterns similar to those used to resolve the same queries in the absence of graduality; and they guarantee the fulfillment of the properties derived from Boolean algebras mentioned in the previous paragraph.

[1]The term *instance* is used here to refer to the set of tuples in a relation at a given moment.

In this work we provide a proposal for resolving queries based on fuzzy grouping, usually known in the literature as *Fuzzy Group By Queries* [BP10], using the alternative approach of RL-instances. To do that, as an additional contribution of this paper, a method is proposed to derive RL-partitions from a fuzzy partition in the Ruspini sense.

As we will see, the results obtained show that this type of queries can be resolved on conventional relational systems using standard SQL query patterns, managing graduality internally by means of RL-instances, while interacting with the user thanks to the well known setting of fuzzy sets.

2 Some preliminary knowledge

2.1 Group by queries

The main capability of SQL is querying, that is, searching for data in the database. Its `SELECT` statement incorporates an important range of tools to carry out this querying process through different clauses.

Its basic `SELECT+FROM+WHERE` form allows us to indicate the relations we want to work with in the `FROM` clause, to conveniently filter their tuples through conditions expressed in the `WHERE` clause, and to determine the subset of columns that we want to be shown through the `SELECT` clause that opens the sentence.

ID	NAME	POSITION	AGE	SALARY
17	Smith	engineer	51	65000
76	Martin	engineer	40	45000
26	Jones	secretary	24	19000
12	Green	technician	39	32000
19	Duncan	clerk	28	24000
8	Brown	manager	54	57000
31	Harris	technician	29	18000
9	Davis	janitor	61	15000
44	Howard	manager	22	45000
23	Lewis	engineer	62	59000

Figure 1. Relation `Employees`

Let us consider the relation of Figure 1 (taken from [BP10], with minor changes). A basic SQL query to retrieve *the name and position of those employees with a salary over 30000* is the following one, with the result shown in Figure 2a:

```
SELECT name, position
FROM employees
WHERE salary > 30000;
```

Additionally to these basic clauses, the `GROUP BY` clause allows us to partition the resulting tuples based on equality in the value of one or more of their attributes. The user can additionally calculate aggregated values relative to each of the generated sets of tuples thanks to the use of aggregate functions such as `COUNT`, `SUM`, or `AVG`, among others.

For example, we can easily ask for a listing with the *number of employees by position* as follows, with result in Figure 2b:

```
SELECT position, count(*)
FROM employees
GROUP BY position;
```

NAME	POSITION
Smith	engineer
Martin	engineer
Green	technician
Brown	manager
Howard	manager
Lewis	engineer

a.

POSITION	COUNT(*)
engineer	3
secretary	1
technician	2
clerk	1
manager	2
janitor	1

b.

Figure 2. Employees with a salary over 30000 (a) and number of employees per position (b)

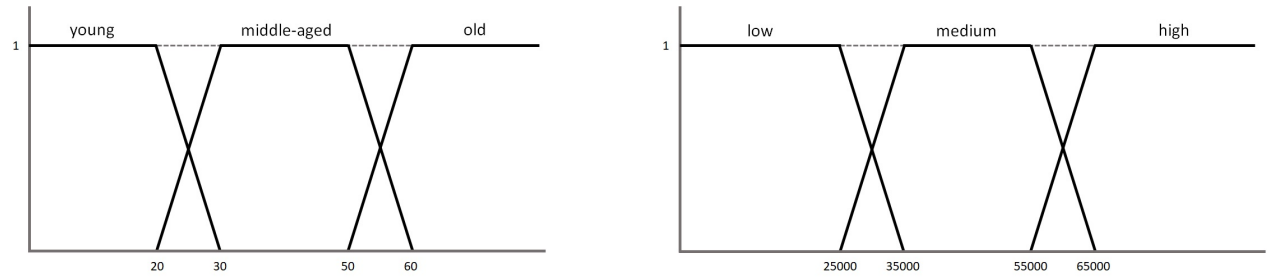
POSITION	COUNT(*)
engineer	2
technician	1
manager	2

a.

POSITION	COUNT(*)
engineer	3
manager	2

b.

Figure 3. Number of employees with salary greater than 30000 per position (a) and number of employees per position where the average salary is greater than 30000 (b)



a. Age partition

b. Salary partition

Figure 4. Fuzzy partitions for age and salary.

The `WHERE` clause is still available to filter tuples before grouping, as in the following query that shows the number of employees with a salary greater than 30000 per position (with result in Figure 3a):

```
SELECT position, count(*)
FROM employees
WHERE salary > 30000
GROUP BY position;
```

Finally, the `HAVING` clause permits filtering the generated sets of tuples according to user preferences. For example, if we want to obtain the number of employees in those groups where average salary is greater than 30000, we can use the following sentence (with result in Figure 3b):

```
SELECT position, count(*)
FROM employees
GROUP BY position
HAVING avg(salary)>30000;
```

2.2 Fuzzy *group by* queries

SQL *group by* queries are a particularly useful operation in data analysis applications to help with decision making and, thus, they have attracted the attention of the fuzzy relational database community.

To illustrate the question, let us complete the example of Figure 1 with the fuzzy partitions on age and salary shown in Figure 4 and with a couple of queries also taken from [BP10].

A fuzzy *group by* query is a query where the partition indicated in the `GROUP BY` clause is fuzzy. In [BP10] the following example is provided: *Find the number of employees with salary greater than 30000 by fuzzy age.*

To resolve this type of query, it is necessary to manage the graduality generated by the fuzzy partitions of the `GROUP BY` clause. The problem becomes more complicated if, in addition, we have to manage graduality in the set of tuples on which the grouping has to be carried out, as in the following example (also taken from [BP10]): *Find the number of employees with medium salary by fuzzy age.*

Proposals relative to the resolution of this type of queries can be found, for example, in [BP10; BPS10; Now18; ZK20]. Additionally, different contributions in relation to the extension of aggregation functions in a fuzzy environment (with special attention to tuple counting) are available in the literature (see for example [BL01; BL08; BLP03; BPL01; Mar+01]).

However, as we said in the introduction, these types of proposals require the use of systems developed from scratch and/or query patterns that are far from those that are usually employed in SQL when solving this type of queries.

The use of RL-instances to solve fuzzy queries has recently appeared in the literature to overcome these drawbacks and additionally provide greater consistency with the mentioned properties derived from Boolean algebras [CMS22]. Within this approach there are already proposals to address fuzzy counting queries [MRS24], although only in the setting of crisp grouping. The case of queries with a fuzzy grouping criterion via RL-instances has not yet been addressed in the literature.

2.3 Representation by levels

Representation by Levels (RL) is an alternative proposal to Fuzzy Set Theory for managing graduality [S+12]. This type of representation makes it possible to handle graduality in any type of object, particularly proposing the notion of RL-set to handle graduality in sets.

A RL-set is a mapping $\rho : \Lambda \to \{0,1\}^X$, where $\Lambda = \{\alpha_1, \dots, \alpha_m\} \subset (0,1]$ with $1 = \alpha_1 > \alpha_2 > \cdots > \alpha_m > \alpha_{m+1} = 0$. The mapping can be extended to the whole $(0,1]$ interval by considering $\rho(\alpha) = \rho(\alpha_i), \forall \alpha_i > \alpha > \alpha_{i+1}$.

Each value in Λ represents a level of *relaxation* or *tolerance* in the definition of the concept or property described by the set, with 1 and 0 standing for the maximum and minimum restrictions, respectively. These values are usually taken equidistributed and in a number based on the desired precision in the system.

It is important to remark that the definition of RL-set does not impose that $\alpha_i < \alpha_j \Rightarrow \rho(\alpha_j) \subseteq \rho(\alpha_i)$, as it is the case of α-cuts in fuzzy sets.

Any operation on conventional sets can be extended to operate on RL-sets by performing the operation on the conventional sets $\rho(\alpha)$ corresponding to each of the levels independently, considering as levels the union of the levels of all the RL-sets involved in the operation. Table 1 shows two example RL-sets A and B defined on a set $O = \{o_1, \dots, o_5\}$ and some operations on them. A and B correspond to the α-cuts of fuzzy sets $1/o_2 + 1/o_3 + 0.5/o_4$ and $1/o_1 + 0.6/o_2 + 0.2/o_3 + 0.2/o_4$, respectively. Note that $\Lambda_A = \{1, 0.5\}$ and $\Lambda_B = \{1, 0.6, 0.2\}$, hence the operations are performed in the set of levels$\{1, 0.6, 0.5, 0.2\}$, using the extensions of RL-sets A and B explained in the second paragraph of this section.

Table 1. Two RL-sets and operations on them.

α	$\rho_A(\alpha)$	$\rho_B(\alpha)$	$\rho_{\overline{B}}(\alpha)$	$\rho_{(A \cap \overline{B})}(\alpha)$
1	$\{o_2, o_3\}$	$\{o_1\}$	$\{o_2, o_3, o_4, o_5\}$	$\{o_2, o_3\}$
0.6	$\{o_2, o_3\}$	$\{o_1, o_2\}$	$\{o_3, o_4, o_5\}$	$\{o_3\}$
0.5	$\{o_2, o_3, o_4\}$	$\{o_1, o_2\}$	$\{o_3, o_4, o_5\}$	$\{o_3, o_4\}$
0.2	$\{o_2, o_3, o_4\}$	$\{o_1, o_2, o_3, o_4\}$	$\{o_5\}$	$\emptyset$

Fuzzy sets and RL-sets can complement each other when developing systems. Thus, a fuzzy set can be obtained from an RL-set, by considering that the degree of membership of a given element to the fuzzy set is the probability that the element is at a level of the RL-set taken at random. This can be seen as a "summary of membership" of objects to the RL. As an example, the fuzzy set summarizing membership to the RL $(A \cap \overline{B})$ in Table 1 is $0.4/o_2 + 0.8/o_3 + 0.3/o_4$. In turn, a fuzzy set can be transformed into a RL-set in different ways (for instance by taking α-cuts, as in the case of A and B in Table 1; we will see examples in Section 3). Note that the same fuzzy set can be obtained for different RL-sets.

2.4 Queries based on RL-instances

Representation by levels have been brought to the world of flexible querying on relational databases through the concept of RL-instance [CMS22].

An RL-instance is a pair (Λ_r, ρ_r), where Λ_r is a set of levels and $\rho_r : \Lambda_r \to I_R$ is a function that assigns a specific instance to each level in Λ_r. That is, an RL-instance is nothing more than an assignment of instances to levels.

In [CMS22] an extension of the operations of relational algebra is proposed to operate with RL-instances. In this work authors show how to solve relational algebra queries that involve fuzzy restrictions by obtaining RL-instances induced by the restrictions and operating conventionally with these resulting RL-instances. The RL-instance obtained

in this process can be transformed into a fuzzy set of tuples (a fuzzy instance) to present to the user the result of the query, according to what is indicated in the last paragraph of the previous section.

3 Management of fuzzy partitions

In order to solve fuzzy *group by* queries through the use of RL-instances we first have to see how to represent the fuzzy partitioning criterion by levels.

According to Ruspini [Rus69], a fuzzy partition of a crisp set $O \neq \emptyset$ is a collection of non-empty fuzzy sets $\mathcal{G} = \{G_1, \dots, G_n\}$ with $n \geq 1$ such that $\forall o \in O$ it is $\sum_{i=1}^{n} G_i(o) = 1$.

On its turn, according to the philosophy of Representations by Levels, a RL-partition of O is an assignment of crisp partitions to levels in $(0, 1]$.

In order to obtain a RL-partition from a fuzzy partition in Ruspini's sense, let us first introduce two special RL-sets defined for any fuzzy set.

3.1 Positive and negative RL-set for a fuzzy set

In Section 2.3, we have seen that different RL-sets may yield the same fuzzy set. Among such RL-sets, we can highlight two of them in which the crisp sets are nested with respect to levels. Let μ_F be a fuzzy set. Then:

- The *positive* RL-set for μ_F, with assignment function denoted by ρ_F^+, corresponds to the usual α-cut representation of fuzzy sets. For every $\alpha \in (0, 1]$, the positive version assigns to α the α-cut of μ_F. That is,

$$\rho_F^+(\alpha) = \{o \in O \text{ s.t. } \mu_F(o) \geq \alpha\}. \tag{1}$$

- The *negative* RL-set for μ_F, with assignment function denoted by ρ_F^-, assigns to every $\alpha \in (0, 1]$ the strong $(1 - \alpha)$-cut of μ_F. That is,

$$\rho_F^-(\alpha) = \{o \in O \text{ s.t. } \mu_F(o) > (1 - \alpha)\}. \tag{2}$$

It is easy to see that the range of ρ_F^+ and ρ_F^- is the same collection of sets, but nested in different order and assigned to different levels in general. Specifically, for every $\alpha, \beta \in (0, 1]$ with $\alpha > \beta$ it is $\rho_F^+(\alpha) \subseteq \rho_F^+(\beta)$ whilst $\rho_F^-(\beta) \subseteq \rho_F^-(\alpha)$.

3.2 From fuzzy partitions to RL-partitions

Let us now see how an RL-partition can be obtained from a fuzzy partition. We shall illustrate our idea with the simplest case: Let $\mathcal{G} = \{G_1, G_2\}$ be a fuzzy partition of a set O. We can obtain a RL-partition with assignment function $\rho_{\mathcal{G}}^a$ as follows:

$$\rho_{\mathcal{G}}^a(\alpha) = \{\rho_{G_1}^+(\alpha), \rho_{G_2}^-(\alpha)\}. \tag{3}$$

As an example, consider $O = \{o_1, \dots, o_5\}$ and let $G_1 = 1/o_1, 0.7/o_2, 0.5/o_3, 0.2/o_4$ and $G_2 = 0.3/o_2, 0.5/o_3, 0.8/o_4, 1/o_5$. Table 2 shows the resulting RL-partition.

However, this is not the only possibility. For the case of two fuzzy sets we can also consider a RL-partition with the following alternative assignment function $\rho_{\mathcal{G}}^b$:

$$\rho_{\mathcal{G}}^b(\alpha) = \{\rho_{G_1}^-(\alpha), \rho_{G_2}^+(\alpha)\}. \tag{4}$$

Table 2. Example: 2 fuzzy sets

α	$\rho_{G_1}^+(\alpha)$	$\rho_{G_2}^-(\alpha)$
1	$\{o_1\}$	$\{o_2, o_3, o_4, o_5\}$
0.8	$\{o_1\}$	$\{o_2, o_3, o_4, o_5\}$
0.7	$\{o_1, o_2\}$	$\{o_3, o_4, o_5\}$
0.5	$\{o_1, o_2, o_3\}$	$\{o_4, o_5\}$
0.3	$\{o_1, o_2, o_3\}$	$\{o_4, o_5\}$
0.2	$\{o_1, o_2, o_3, o_4\}$	$\{o_5\}$

Table 3. Example: 2 fuzzy sets, negative first

α	$\rho^{-}_{G_1}(\alpha)$	$\rho^{+}_{G_2}(\alpha)$
1	$\{o_1, o_2, o_3, o_4\}$	$\{o_5\}$
0.8	$\{o_1, o_2, o_3\}$	$\{o_4, o_5\}$
0.7	$\{o_1, o_2, o_3\}$	$\{o_4, o_5\}$
0.5	$\{o_1, o_2\}$	$\{o_3, o_4, o_5\}$
0.3	$\{o_1\}$	$\{o_2, o_3, o_4, o_5\}$
0.2	$\{o_1\}$	$\{o_2, o_3, o_4, o_5\}$

In this case, the resulting RL-partition is that in Table 3. Note that objects with membership degree 0.5 are always assigned to the 0.5-cut of positive versions. It is also easy to see that for negative versions, level 1 corresponds to the support of the fuzzy set.

Note also that, in order to obtain a RL-partition from a fuzzy partition comprised of two fuzzy sets, we need to incorporate an additional information in order to determine which of the two possible RL-partitions we want to consider. We can see this choice indeed as coming from a provided total ordering $\prec$ of the sets being considered, so that the first set in the total order is considered to be positive and the second, negative. This way, when we consider $G_1 \prec G_2$ we have the solution in Table 2, and we have that in Table 3 when $G_2 \prec G_1$.

The case of two fuzzy sets we have just illustrated is a particular case of the following more general case:

Theorem 3.1
Let $\mathcal{G} = \{G_1, \ldots, G_n\}$ be a fuzzy partition in Ruspini's sense of a non-empty set of objects O. Let $\prec$ be a total order defined among sets of $\mathcal{G}$ as $G_1 \prec G_2 \prec \cdots \prec G_n$ satisfying the following condition: if $support(G_i) \cap support(G_j) \neq \emptyset$ for some $G_i, G_j \in \mathcal{G}$ then it is $i = j + 1$ or $i = j - 1$. Then, the following are two RL-partitions of O:

1. $\rho^{a}_{\mathcal{G}}(\alpha)$ contains the sets $\rho^{+}_{G_i}(\alpha)$ for odd values of i and the sets $\rho^{-}_{G_i}(\alpha)$ for even values of i.
2. $\rho^{b}_{\mathcal{G}}(\alpha)$ contains the sets $\rho^{-}_{G_i}(\alpha)$ for odd values of i and the sets $\rho^{+}_{G_i}(\alpha)$ for even values of i.

As an example we can consider the typical fuzzy partitions of a numerical domain using trapezoidal functions that only have non-empty intersection with the immediate trapezoids at their left and right. Then, a total order can be defined by considering the trapezoids from left to right with two possible RL-partitions: starting with the positive version or starting with the negative version. The reader can easily see that taking the total order by considering the trapezoids from right to left will lead to the same two RL-partitions. In this work, without loss of generality, we will take the order from left to right starting with the positive version.

4 Patterns for fuzzy group by queries through RL-instances

We have seen in the previous section how we can turn a fuzzy partition in the Ruspini sense into a crisp partition at each of the levels. Let us now see how we can resolve fuzzy *group by* queries through standard SQL thanks to this approach.

To do this, the following has been created in the database:

- A relation `lambda` with a single attribute `alpha` and as many tuples as levels we want to use (in our case, we have worked with 10 levels equidistributed in (0,1]).
- A `fuzzy_trapezoid` type to handle labels with trapezoidal membership function, whose objects are capable of indicating wether they are compatible with a value of the base domain at a given alpha level or not. This is done by the `matches` method.
- A `fuzzy_partition` type to handle fuzzy partitions, whose objects are capable of indicating the label that corresponds to a value of the base domain at a given alpha level (using the approach explained in Section 3). They do this thanks to the `segment` method.

Both types incorporate constructors to build new objects from a catalog of labels and partitions previously created by the user in the database. It is not necessary to add anything else to the conventional relational system to be able to resolve queries like the ones we will see below.

4.1 Fuzzy *group by* on crisp instances

First, let us focus on a query on a crisp instance with a fuzzy grouping criterion. For example, the query *Find the number of employees with salary greater than 30000 by fuzzy age* that we mentioned in Section 2.2.

Following the conventional SQL pattern for this type of queries described in Section 2.1, the query is as follows (with the result shown in Figure 5a):

```
SELECT alpha, fuzzy_partition('fuzzy_age').segment(age, alpha) f_age, COUNT(*) cnt
FROM employees, lambda
WHERE salary >30000
GROUP BY alpha, fuzzy_partition('fuzzy_age').segment(age, alpha)
ORDER BY alpha DESC, f_age;
```

Note that, in order to work with RL-instances, a cartesian product of `employees` and `lambda` is carried out. A `fuzzy_partition` object with the fuzzy partition for age is responsible for indicating the age label that corresponds to each tuple at each level.

If, instead of an RL-instance, we want to display a fuzzy instance as result, it is enough to adapt the query in the following way (with the result shown in Figure 5b), proceeding as indicated in Section 2.3:

```
WITH base_query AS (
    SELECT alpha, fuzzy_partition('fuzzy_age').segment(age, alpha) f_age, COUNT(*) cnt
    FROM employees, lambda
    WHERE salary >30000
    GROUP BY alpha, fuzzy_partition('fuzzy_age').segment(age, alpha))
SELECT COUNT(alpha)/10 mu, f_age, cnt
FROM base_query
GROUP BY f_age, cnt
ORDER BY mu DESC, f_age;
```

In the *fuzzy* result shown in Figure 5b, a tuple with `cnt` equal to 0 is missing for some of the `f_age` values. The membership of this missing tuple regarding the 0 value for `COUNT` is the result of subtracting the sum of the other degrees for the label from 1. In the example, only a tuple is missing: <0.2, young, 0>. This behavior is similar to that of conventional count queries, in which tuples corresponding to the values of the *group by* attribute that do not appear in the table are missing in the result.

Note that the obtained result, that summarizes the RL result, takes the form of a probability distribution over the naturals for every fuzzy label. Following the idea of RLs, for every fuzzy label, the values in $[0, 1]$ associated to each natural number represent the probability that, in a level taken at random, the count of tuples in the level is precisely that number. In our opinion, this approach improves the interpretability of the result compared to other approaches that offer a single aggregate as result (usually a real number obtained by aggregating memberships or their combination with other numerical values). The same idea can be employed when dealing with other aggregate functions.

For the sake of space, and given that the sentence that obtains the RL-instance is shown as `base_query` in the sentence that obtains the fuzzy instance, in the following examples we will only show the code of the latter.

4.2 Fuzzy *group by* on fuzzy instances

Resolving fuzzy *group by* queries using RL-instances allows us to easily perform the queries on fuzzy instances instead of crisp instances, without having to vary the SQL pattern. This is achieved by taking advantage of the previously mentioned `matches` method.

For example, let us consider the second query we mentioned in Section 2.2: *Find the number of employees with medium salary by fuzzy age.* The SQL query for this example has the same code as the previous example, just changing the `WHERE` clause to control the compatibility of the salary with the label `medium` at each level (the result is shown in Figure 6):

```
WITH base_query AS (
    SELECT alpha, fuzzy_partition('fuzzy_age').segment(age, alpha) f_age, COUNT(*) cnt
    FROM employees, lambda
    WHERE Fuzzy_Trapezoid('medium').matches(salary, alpha)='T'
    GROUP BY alpha, fuzzy_partition('fuzzy_age').segment(age, alpha))
```

α	F_AGE	CNT
1.0	middle-aged	5
1.0	old	1
0.9	middle-aged	5
0.9	old	1
0.8	middle-aged	4
0.8	old	1
0.8	young	1
0.7	middle-aged	4
0.7	old	1
0.7	young	1
0.6	middle-aged	4
0.6	old	1
0.6	young	1
0.5	middle-aged	4
0.5	old	1
0.5	young	1
0.4	middle-aged	3
0.4	old	2
0.4	young	1
0.3	middle-aged	3
0.3	old	2
0.3	young	1
0.2	middle-aged	3
0.2	old	2
0.2	young	1
0.1	middle-aged	2
0.1	old	3
0.1	young	1

a. RL-instance

μ	F_AGE	CNT
0.4	middle-aged	4
0.3	middle-aged	3
0.2	middle-aged	5
0.1	middle-aged	2
0.6	old	1
0.3	old	2
0.1	old	3
0.8	young	1

b. Fuzzy instance

Figure 5. Number of employees with salary greater than 30000 per fuzzy age.

```
SELECT COUNT(alpha)/10 mu, f_age, cnt
FROM base_query
GROUP BY f_age, cnt
ORDER BY f_age, mu DESC;
```

4.3 Filtering groups

Resolving fuzzy *group by* queries using RL-instances also allows the user to impose `HAVING` conditions on the generated groups, regardless of whether these conditions are fuzzy or not.

For example, we can obtain the *age labels for which the average salary of engineers is high* as follows [BP10](with the result shown in Figure 7):

```
WITH base_query AS (
    SELECT alpha, fuzzy_partition('fuzzy_age').segment(age, alpha) f_age
    FROM employees, lambda
    WHERE position = 'engineer'
    GROUP BY alpha, fuzzy_partition('fuzzy_age').segment(age, alpha)
    HAVING fuzzy_trapezoid('high').matches(AVG(salary), alpha)='T')
SELECT COUNT(alpha)/10 mu, f_age
FROM base_query
GROUP BY f_age
ORDER BY f_age, mu DESC;
```

5 Conclusions

In this work we have addressed the resolution of fuzzy *group by* queries through the use of RL-instances. To do this, we have provided a method to convert fuzzy partitions in the Ruspini sense into RL-partitions, so that queries can be resolved with conventional query patterns of SQL. Our results allow this type of capabilities to be incorporated in a very simple way in any SQL compliant system, smoothing the user's learning curve.

α	F_AGE	CNT
1.0	middle-aged	2
0.9	middle-aged	2
0.8	middle-aged	2
0.8	young	1
0.7	middle-aged	3
0.7	young	1
0.6	middle-aged	3
0.6	old	1
0.6	young	1
0.5	middle-aged	3
0.5	old	1
0.5	young	1
0.4	middle-aged	2
0.4	old	2
0.4	young	1
0.3	middle-aged	2
0.3	old	2
0.3	young	1
0.2	middle-aged	2
0.2	old	2
0.2	young	1
0.1	middle-aged	2
0.1	old	2
0.1	young	1

a. RL-instance

μ	F_AGE	CNT
0.7	middle-aged	2
0.3	middle-aged	3
0.4	old	2
0.2	old	1
0.8	young	1

b. Fuzzy instance

Figure 6. Number of employees with medium salary per fuzzy age.

α	F_AGE
0.4	old
0.3	old
0.2	old
0.1	old

a. RL-instance

μ	F_AGE
0.4	old

b. Fuzzy instance

Figure 7. Fuzzy ages where engineers have in average a high salary.

Acknowledgment

This publication is part of the Grant PID2021-126363NB-I00 funded by MICIU/AEI/10.13039/501100011033 and by "ERDF A way of making Europe".

References

[BL01] Patrick Bosc and Ludovic Liétard. "Introducing the aggregate "count" into flexible queries". In: *EUSFLAT 2001, Leicester, United Kingdom, September 5-7, Proceedings*. De Montfort University, Leicester, UK, 2001, pp. 202–205.

[BL08] Patrick Bosc and Ludovic Liétard. "Aggregates Computed over Fuzzy Sets and their Integration into SQLf". In: *Int. J. Uncertain. Fuzziness Knowl. Based Syst.* 16.6 (2008), pp. 761–792.

[BLP03] Patrick Bosc, Ludovic Liétard, and Olivier Pivert. "Sugeno fuzzy integral as a basis for the interpretation of flexible queries involving monotonic aggregates". In: *Inf. Process. Manag.* 39.2 (2003), pp. 287–306.

[BP10] Patrick Bosc and Olivier Pivert. "On a fuzzy group-by clause in SQLf". In: *FUZZ-IEEE 2010, Barcelona, Spain, 18-23 July, Proceedings*. IEEE, 2010, pp. 1–6.

[BPL01] Patrick Bosc, Olivier Pivert, and Ludovic Liétard. "Aggregate Operators in Database Flexible Querying". In: *FUZZ-IEEE 2001, Melbourne, Australia, December 2-5, Proceedings*. 2001, pp. 1231–1234.

[BPS10] Patrick Bosc, Olivier Pivert, and Grégory Smits. "On a Fuzzy Group-By and Its Use for Fuzzy Association Rule Mining". In: *ADBIS 2010, Novi Sad, Serbia, September 20-24, Proceedings*. Ed. by Barbara Catania, Mirjana Ivanovic, and Bernhard Thalheim. Vol. 6295. Lecture Notes in Computer Science. Springer, 2010, pp. 88–102.

[CMS22] Patricia Córdoba-Hidalgo, Nicolás Marín, and Daniel Sánchez. "RL-instances: An alternative to conjunctive fuzzy sets of tuples for flexible querying in relational databases". In: *Fuzzy Sets Syst.* 445 (2022), pp. 184–206.

[DP80] Didier Dubois and Henri Prade. "New Results about Properties and Semantics of Fuzzy Set-Theoretic Operators". In: *Fuzzy Sets: Theory and Applications to Policy Analysis and Information Systems*. Ed. by Paul P. Wang and S. K. Chang. Boston, MA: Springer US, 1980, pp. 59–75.

[Gal08] José Galindo, ed. *Handbook of Research on Fuzzy Information Processing in Databases*. IGI Global, 2008.

[KZD15] Janusz Kacprzyk, Slawomir Zadrozny, and Guy De Tré. "Fuzziness in database management systems: Half a century of developments and future prospects". In: *Fuzzy Sets Syst.* 281 (2015), pp. 300–307.

[MRS24] Nicolás Marín, Gustavo Rivas-Gervilla, and Daniel Sánchez. "Count-based Flexible Queries Through RL-Instances". In: *FUZZ-IEEE 2024, Yokohama, Japan, June 30 - July 5, Proceedings*. IEEE, 2024, pp. 1–8.

[Mar+01] N. Marín, D. Sánchez, J.M. Serrano, and M.A. Vila. "Problems of Fuzzy Queries Involving Aggregation Functions: The 'Select Count' Case". In: *IFSA/NAFIPS 2001, Vancouver, Canada, 25-28 July, Proceedings*. 2001, pp. 2132–2137.

[Now18] Grzegorz Nowakowski. "Fuzzy queries on relational databases". In: *2018 International Interdisciplinary PhD Workshop (IIPhDW)*. 2018, pp. 293–299.

[Now19] Grzegorz Nowakowski. "Methodology of Transformation of Fuzzy Queries into Queries in the SQL Standard". In: *IDAACS 2019, Metz, France, September 18-21, Proceedings*. IEEE, 2019, pp. 674–679.

[PS21] Olivier Pivert and Grégory Smits. "Fuzzy Extensions of Databases". In: *Fuzzy Approaches for Soft Computing and Approximate Reasoning: Theories and Applications - Dedicated to Bernadette Bouchon-Meunier*. Ed. by Marie-Jeanne Lesot and Christophe Marsala. Vol. 394. Studies in Fuzziness and Soft Computing. Springer, 2021, pp. 191–200.

[PZ14] Olivier Pivert and Slawomir Zadrozny, eds. *Flexible Approaches in Data, Information and Knowledge Management*. Vol. 497. Studies in Computational Intelligence. Springer, 2014.

[Rus69] Enrique H Ruspini. "A new approach to clustering". In: *Information and control* 15.1 (1969), pp. 22–32.

[S+12] D. Sánchez, M. Delgado, M.A. Vila, and J. Chamorro-Martínez. "On a Non-nested Level-Based Representation of Fuzziness". In: *Fuzzy Sets and Systems* 192.1 (2012), pp. 159–175.

[TZ15] Guy De Tré and Slawomir Zadrozny. "Soft Computing in Database and Information Management". In: *Springer Handbook of Computational Intelligence*. Ed. by Janusz Kacprzyk and Witold Pedrycz. Springer Handbooks. Springer, 2015, pp. 295–312.

[ZK20] Slawomir Zadrozny and Janusz Kacprzyk. "Fuzzy Analytical Queries: A New Approach to Flexible Fuzzy Queries". In: *FUZZ-IEEE 2020, Glasgow, UK, July 19-24, Proceedings*. IEEE, 2020, pp. 1–8.