ppi 201502ZU4659
This scientific digital publication is continuance of the printed journal
p-ISSN 0254-0770 / e-ISSN 2477-9377 / Legal Deposit pp 197802ZU38

La Ciega Historical building. LUZ first headquarter

REVISTA TÉCNICA
DE LA FACULTAD DE INGENIERÍA

"Post nubila phoebus"
"After the clouds, the sun"

LUZ in its 130th anniversary
Established since 1891

An international refereed Journal indexed by:

- SCOPUS
- SCIELO
- LATINDEX
- DOAJ
- MIAR
- REDIB
- AEROSPACE DATABASE
- CIVIL ENGINEERING ABTRACTS
- METADEX
- COMMUNICATION ABSTRACTS
- ZENTRALBLATT MATH, ZBMATH
- ACTUALIDAD IBEROAMERICANA
- BIBLAT
- PERIODICA
- REVENCYT

REVISTA TÉCNICA
DE LA FACULTAD DE INGENIERÍA

UNIVERSIDAD DEL ZULIA

VOLUME 44 JANUARY - APRIL 2021 NUMBER 1

Rev. Téc. Ing. Univ. Zulia. Vol. 44, No. 1, January-April, 2021, 36-43

An Implementation for SQL Fuzzy Grouping

***Ana Isabel Aguilera Faraco*[*1], *Marlene Goncalves Da Silva*[2]**

[1]Escuela de Ingeniería Informática, Facultad de Ingeniería, Universidad de Valparaíso, Valparaíso, C.P. 2340000, Chile
[2]Departamento de Computación y Tecnología de la Información, Universidad Simón Bolívar, Caracas, Venezuela, Apartado 89000, Caracas, Venezuela.
*Corresponding author: ana.aguilera@uv.cl
https://doi.org/10.22209/rt.v44n1a05
Received: 13 de abril de 2020 | Accepted: 20 de octubre de 2020 | Available: 01 de enero de 2021

Abstract

Relational DataBasemanagement systems (DBMS) have a great utility in the efficient storage of large data volumes. Also, some DBMS extensions based on fuzzy logic have been proposed to improve the expressiveness of query languages. Among which SQLf is an extension of SQL that supports fuzzy conditions. Separately, the Group-By is a database operator widely used in data analysis and decision support systems. In many cases, it seems useful to group values according to their similarity to a certain concept rather than establishing grouping on the basis of equal values. In this context, a new SQLf structure called Fuzzy Group By (FGB) has been proposed to support a grouping based on fuzzy partitions. In this work, we incorporated the fuzzy grouping in PostgreSQLf, which is an extension of the PostgreSQL DBMS for the handling of fuzzy queries using the SQLf language on the basis of a tight coupled architecture, i.e., directly into the DBMS. We have proposed an algorithm based on a *hash* to evaluate the FGB operator and also empirically assessed the performance of PostgreSQLf over the TPC Benchmark™ -H (TPC-H).

Keywords: Fuzzy Group By; PostgreSQLf; tight coupled architecture

Una Implementación para el Agrupamiento Difuso en SQL

Resumen

Los sistemas de gestión de bases de datos (SGBD) relacionales tienen una gran utilidad en el almacenamiento eficiente de grandes volúmenes de datos. En este sentido, se han propuesto algunas extensiones de los SGBD basadas en la lógica difusa, para mejorar la expresividad de los lenguajes de consulta, entre ellos, el lenguaje SQLf (extensión de SQL que soporta condiciones difusas). Por otra parte, el Group-By es un operador de base de datos ampliamente utilizado en el análisis de datos y en los sistemas de apoyo a la toma de decisiones. En muchos casos, parece útil agrupar los valores según su similitud con un determinado concepto en lugar de establecer la agrupación sobre la base de valores iguales. En este contexto, se ha propuesto una nueva estructura de SQLf denominada Fuzzy Group By (FGB), para apoyar una agrupación basada en particiones difusas. En este trabajo, se incorporó la agrupación difusa en PostgreSQLf, que es una extensión del SGBD PostgreSQL, para el manejo de consultas difusas utilizando el lenguaje SQLf con una arquitectura fuertemente acoplada (directamente en el SGBD). Se pronone un algoritmo basado en un *hash* para evaluar el operador FGB y también se evalúa empíricamente el rendimiento de PostgreSQLf sobre el Benchmark™ TPC-H.

Palabras clave: Fuzzy Group By; PostgreSQLf; arquitectura fuertemente acoplada.

Introduction

Despite the dizzying development of database technology, common DataBase management systems (DBMSs) do not allow the expression of gradual users' requirements because they suffer from the problem of rigidity [1], [2], [3]. In a classic system, users' requirements must be expressed in a precise manner. In this sense, the rigidity of classical systems has two main consequences [4]. There is no discrimination of responses according to users' preferences and the borderline responses can leave out results. Fuzzy logic offers new tools for accessing and processing data that may have applicability in systems where users' requirements are not precise by nature [5], [6], [7].

The standard Group-By operator has great importance for data warehouses and analysis techniques such as OLAP and data mining, and it has relatively good runtime and scalability properties. Even though, the semantics of Group-By is simple, it is limited to equality (all tuples in a group have exactly the same values as the grouping attributes). To overcome these shortcomings, Bosc and Pivert [8] propose to extend the Group-By clause in SQLf by means of the Fuzzy Group-By (FGB) clause, which allows grouping based on predefined fuzzy partitions in the domain attributes instead of data equality. A Group-By clause in SQL builds a partition based on the (atomic) values of the attributes specified in that clause, e.g., "GROUP BY A" constructs a partition where every group is associated with a value of A present in the relationship. Bosc and Pivert's idea is to extend this mechanism in order to build partitions in terms of intervals or fuzzy sets of values. In addition, they add a variant of the count aggregation function called *count-rel*, which calculates the relative cardinality (e.g., the average satisfaction degree) associated with a group.

Emerging applications such as biological databases and data streaming require identification of groups of approximate values. In addition, business applications with large amounts of data, can tremendously benefit from SQLf statements that identify groups of similar values. The implementation of fuzzy grouping within a database engine can have the advantage that the execution time of Fuzzy Group-By is comparable to the conventional Group-By. Thus, this work comprises the development of an extension of the PostgreSQL DBMS for the management of fuzzy grouping on the basis of a tight coupled architecture. In a tight coupled architecture [9], [10], [8], all tasks, components and functionalities corresponding to the database paradigm to be integrated are part of the respective DBMS as a primitive operation. The main advantage of this architecture is that all scalability and performance problems that may present themselves in other types of architectures are solved.

Motivating example

Consider the Billboard Chart data shown in Table 1, where an item is characterized by a title, year, artist and sales in millions. Also, consider a user's query to determine the mean of sales by decade (sixties, seventies, eighties, nineties, etc.). In SQL, this query can be expressed as [8], [11]: SELECT label(year), avg(sales) FROM billboard_chart GROUP BY label(year) USING p(year) = {[1960, 1969], [1970, 1979], [1980, 1989], [1990, 1999], [2000, 2009], [2010,2019]}; The result of this query is presented in Table 2.

Table 1. The billboard chart data.

Title	Year	Artist	Sales (millions)
Can't Help Falling In Love	1962	Elvis Presley	28
Carnegie Hall Concert	1966	Buck Owens	54
Aretha Franklin: Soul '69	1969	Aretha Franklin	32
Something Better To Do	1975	Olivia Newton-John	22
Thriller	1983	Michael Jackson	65
This Is The Time	1987	Billy Joel	12
Ballerina Girl	1987	Lionel Richie	53
My Heart Will Go On	1998	Celine Dion	8
Hard Candy	2008	Madonna	34
No Line On The Horizon	2009	U2	31
Someone Like You	2011	Adele	41
Love Yourself	2016	Justin Bieber	23
Cozy Little Christmas	2018	Katy Perry	12

Table 2. Mean sales of titles by year range.

***Label* (year)**	[1960, 1969]	[1970, 1979]	[1980, 1989]	[1990, 1999]	[2000, 2009]	[2010, 2019]
mean	38.00	22.00	43.33	8.00	32.50	25.33

A classical query with a GROUP BY clause is as follows [3], [9]: SELECT label(A) [, agg, ...] FROM R WHERE ψ GROUP BY label (A) USING p (a)={L_1, ..., L_n}; Where p (A) is a partition defined in the domain A, label (A) denotes a tag L_i of p (A), and ψ is a Boolean or fuzzy condition. If the WHERE clause has a fuzzy condition, COUNT can be only used as an aggregation function in the clause since the difficulty of defining other aggregation functions on fuzzy sets. For a given Li belonging to p(A) and a relationship r, count(Li) is computed. Additionally, in this work, we use a count variant named *count-rel* which calculates relative cardinality associated to a group, e.g. average degree of satisfaction [8], [11]:

$$count(Li) = \sum_{t \in r \wedge t.A \epsilon Li} \mu_{\psi}(t) \quad count - rel(Li) = \frac{\sum_{t \epsilon r \wedge t.A \epsilon Li} \mu_{\psi}(t)}{|\{t \epsilon r \mid t.A \epsilon Li\}|} \quad (1)$$

To illustrate the *count-rel* use, consider the following query: **SELECT label(year), count, count-rel FROM billboard_chart WHERE sales=medium GROUP BY label(year) USING p(year) = {[1960, 1969], [1970, 1979], [1980, 1989], [1990, 1999], [2000, 2009], [2010,2019]};** Where the medium sales term is defined by means of the trapezium shown in Figure 1(b), the calculations are in Table 3 and the results of the query are in Table 4; *n* represents $\{t \epsilon r \mid t.A \epsilon Li\}$. Partitioned queries may be extended by means of fuzzy logic. To exemplify fuzzy partitioned queries, consider the trapeziums in Figure 1 and the following query that determines the volume of sales for all titles newer than 1990 for each sales class (low, medium, high)**: SELECT label(sales), count FROM billboard_chart WHERE year > 1990 GROUP BY label(sales) USING p(sales) = {low, medium, high}.**

On the basis of data in Table 1, the result of this query is presented in Table 5. That is, for the label(sales)= high the count corresponds to sum of the membership degree of each tuple that has a year greater than 1990, they are: {My Heart Will Go On/0, Hard Candy/0.7, No Line On The Horizon/0.55, Someone Like You/1, Love Yourself/0.15, Cozy Little Christmas/0}, similarly for label(sales)= medium and label(sales)= low.

It is important to note that for those cases where the condition is fuzzy, it is necessary to redefine *count* and *count-rel* as follows [8], [11]:

$$count(Li) = \sum_{t \epsilon r} min\left(\mu_{\psi}(t), \mu_{Li}(t.A)\right) \quad count - rel(Li) = \frac{\sum_{t \epsilon r} min\left(\mu_{\psi}(t), \mu_{Li}(t.A)\right)}{\sum_{t \epsilon r} \mu_{Li}(t.A)} \quad (2)$$

Figure 1. Fuzzy terms defined by trapeziums: (a) low (b) medium (c) high

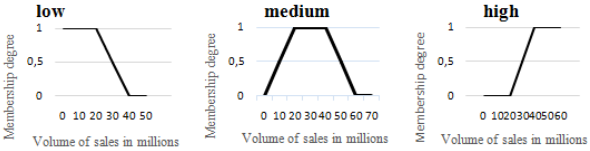

Table 3. Computed satisfaction degrees for the fuzzy partitioned query.

Label (year)	I	Title	sales	$\mu_{\psi}(t)$	$\sum_{i \epsilon n} \mu_{\psi}(t)$	$\frac{\sum_{i \epsilon n} \mu_{\psi}(t)}{\lvert n \rvert}$
[1960-1969]	1	Can't Help Falling In Love	28	1.00	2.08	0.69
	2	Carnegie Hall Concert	54	0.08		
	3	Aretha Franklin: Soul '69	32	1.00		
[1970-1979]	1	Something Better To Do	22	1.00	1.00	1.00
[1980-1989]	1	Thriller	65	0.00	1.15	0.38
	2	This Is The Time	12	0.60		
	3	Ballerina Girl	53	0.55		
[1990-1999]	1	My Heart Will Go On	8	0.45	0.45	0.45
[2000-2009]	1	Hard Candy	34	1.00	2.00	1.00
	2	No Line On The Horizon	31	1.00		
[2010-2019]	1	Someone Like You	41	0.95	2.55	0.85
	2	Love Yourself	23	1.00		
	3	Cozy Little Christmas	12	0.6		

Table 4. Query results using a defined partition on age.

label (ventas)	[1960, 1969]	[1970, 1979]	[1980, 1989]	[1990, 1999]	[2000, 2009]	[2010, 2019]
count	2,08	1,00	1,15	0,45	2,00	2,55
count-rel	0,69	1,00	0,38	0,45	1,00	0,85

Table 5. Query results using a fuzzy partition on sales.

***Label* (sales)**	High	Medium	low
count	2.40	4.95	3.45

Experimental

PostgreSQLf

In this work, we extended PostgreSQL based us on a tightly coupled integration architecture, which takes advantages of the internal characteristics of DBMS [12], [13], [14]. Unfortunately, more effort can be required due to the complexity of open source system and additionally the portability to new version is completely lost. Even if the portability was compromised, our interesting is focused in the performance and to show the feasibility of implementation as a proof of concept. This work is part of another big project called PostgreSQLf started in 2006 [15]. Figure 2 shows the modules of PostgreSQL that we have modified (in blue color). A query executed by PostgreSQL must pass through three modules: i) the Parser that verifies the query validity; ii) the Planner that builds the execution plan for the query; iii) the Executor where each query is finally evaluated.

Catalog

The DBMS catalog is a repository where the metadata of the database schema is stored (information of tables, columns, indexes, operators, aggregate functions, among others). PostgreSQL manages these metadata as system tables. Particularly, in this work, the catalog was extended to add the new aggregation functions of the Fuzzy Grouping as part of the standard PostgreSQL functions.

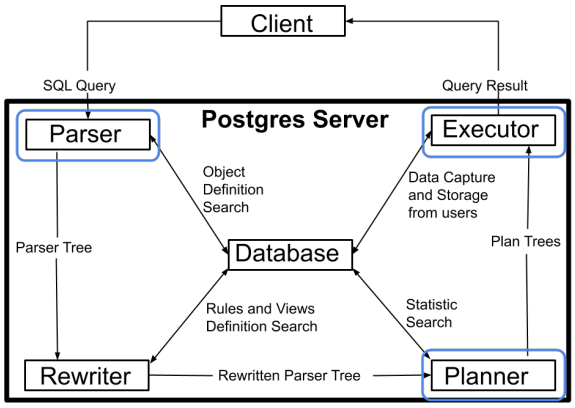

Figure 2. PostgreSQL modules

Parser

The parser comprises two processes: the syntactic and semantic analysis, and the transformation process that takes the query sentence and converts it to data structures operable by PostgreSQL. The query trees are managed by means of linked lists. The lexical analyzer recognizes identifiers, SQL keywords, etc. For queries with Fuzzy Grouping, a new type of node called *A_Partition* was created with all the relevant data of the partition such as its domain, its labels, among others. A pointer (*usingClause*) is added to the *SelectStmt* node in a list of partition nodes.

Planner

The task of this module is to create a fuzzy execution plan. Thus, it combines possible access paths of the relationships (index scan, sequential scan or bitmap index scan) and joins them if it is necessary (nested loop join, hashed join or merge sort join). The task of this module is to estimate the execution costs of each access path and choose the cheapest one. In this work, the received query tree is verified and then a fuzzy plan is created from the query tree. Additional plan nodes are built to calculate the cardinalities necessary for the aggregate functions of the fuzzy grouping.

Executor

This module takes the Planner execution plan and processes it recursively to obtain the required set of rows; it uses a progressive demand pipeline mechanism. Each time a plan-type node is called, it must dispatch one or more rows, or report that it has finished all the required rows. To evaluate the Fuzzy Group By operator, we propose an algorithm based on a *hash*. Each partition represents a bucket (a fuzzy set) and the *hash* function discriminates to which bucket a tuple belongs and its membership degree. Algorithm 1 shows the process of filling the *hash* table for fuzzy grouping.

Algorithm 1. Filling the *hash* table for fuzzy grouping

```
NewTuple ← Ø;
for each Partition in Query do
        for each Label in Partition do
          if Tuple is in Label domain then
                NewTuple ← copyTuple(Tuple); M ← SatisfactionDegreeFGB(Tuple, Label);
                HashTableEntry ← insertInHashTable(NewTuple);
                if Query has count_rel then
                N←LabelCardinality(Label); Advance Aggregates(HashTableEntry, M, N);
        elseAdvanceAggregates(HashTableEntry, M);
```

First, the executor calculates the membership grades for the predicates that represent the partition labels in the fuzzy grouping. For queries with fuzzy or classic grouping, groups are created on the fly based on the labels specified by the user. In the *nodeAgg.c* file of PostgreSQL, the way in which the system groups the tuples must be

modified since for each tuple, the result is grouped by the value of one or more fields normally, i.e., if it is grouped by an age field, then there are tuples that are grouped by age= 23, others by age= 31, etc. In other words, the groups are disjoint sets, where there are no elements found in more than one set (Figure 3a). This classification is done by means of a *Hash* table that is then extracted for each group to apply final calculations as aggregate functions. The difference in fuzzy grouping is that each tuple can belong to one or more groups (Figure 3b), this happens when overlapping tags (classic or fuzzy) are used within the partition. Therefore, for the cases of quantified queries, the *agg_fill_hash_table* function verifies if each tuple in the process belongs to some label of each partition specified in the query. If the tuple belongs to more than one label, it is duplicated and introduced into the *hash* table. If it is a fuzzy partition or there are fuzzy predicates in the query, its membership grade is calculated and used for the calculations of the *count_p* and *count_prel* aggregate functions. For the latter, it will be necessary to calculate the cardinality of each label.

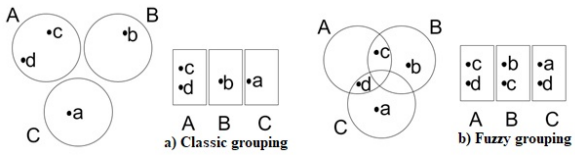

Figure 3. Difference between standard and proposed groupings

The implementation of the function *advanceAgregates* is exactly the same as the default function of PostgreSQL, except that the *count_p aggregate* function is a sum function that adds the membership degrees as parameters and the function *count_prel* is equal to a sum function but adds the division between the membership degree and cardinality passed as parameters.

Experimental study

In this section, we study the performance of fuzzy grouping queries inside PostgreSQL. First, we describe the benchmark, the metrics and implementation details for our experimental study.

Benchmark

TPC-H™ provides a database schema, a data generator, and queries to evaluate the performance of a system under standard conditions. The dataset sizes were 1 and 5 GB. In this experimental study, the tables *part* (200,000/1,000,000 rows), *partsupp* (800,000/4,000,000 rows), and *supplier* (10,000/50,000 rows) were considered. Only the data generator was used because we evaluate our own queries with fuzzy grouping. Thus, 24 queries were defined: i) six queries with Boolean condition and classical partition; ii) six queries with fuzzy condition and classical partition; iii) six queries with Boolean condition and fuzzy partition; iv) six queries with fuzzy condition and fuzzy partition. Also, six equivalent classical queries were defined.

Evaluation metrics

Performance is reported and measured as total execution time (the elapsed time in milliseconds between the submission of a query to PostgreSQL and the delivery of the answers). Time was measured using the SQL EXPLAIN ANALYZE.

Implementation

Experiments were executed using PostgreSQL 8.2 on Ubuntu Desktop 10.10, architecture AMD64, equipped with an Intel Core 2 Duo T6400 and 4 GB RAM.

Results and discussion

Afterwards, exploratory analysis of the data was performed using descriptive statistics. For this purpose, a histogram of time (ms) was plotted in Figure 4. According to the dataset, the highest frequency is in the second interval (between ≈333 and ≈666 ms) and the data distribution resembles a log-normal function (Figure 4 left). As a measure of location, we have the mean, in this case was 1,034.76739 ms. There is also a very high variability of the data since the standard deviation was 959.035129 ms. With respect to measure of dispersion, the difference between the minimum and maximum values was high (101.477 and 3,900.827 ms).

Figure 4 (center) contains a boxplot for the total execution time. Considering the type of query in Figure 4 (right), it showe that the classic Boolean queries have the highest times. Although the classic queries return the same tuples and are grouped in the same way as the extension, they use the UNION operator which generates an additional workload increasing the execution time. Thus, our solution experimentally shows that it is more efficient for grouping queries. Also, there is one outlier which is related to the returned high data volume.

Subsequently, an ANOVA was performed to determine if there was significant difference between the sample means (Table 6). In this regard, with a significance level of 95% (α= 0.05), there was difference when varying the database size or the type of queries. That is, the database volume and the type of queries significantly affect the execution time of the query.

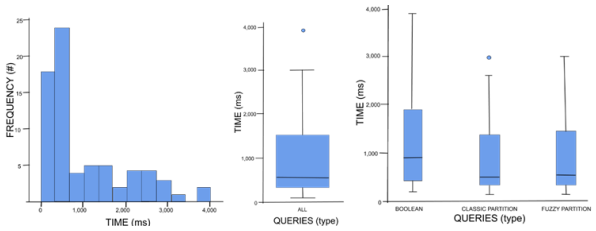

Figure 4. Histogram of total execution time (left). Histogram of total execution time boxplot (center). Histogram of query type boxplot (right)

particular SQL functionality.

Based on the similarity-based-group-by construct, Bosc and Pivert [8] proposed how to introduce the grouping of data in terms of vague concepts (fuzzy predicates) within the SQLf statements [8], [11]. Lastly, some other implementations of fuzzy grouping have been developed. A *windows* program called fuzzy grouping offers three methods of fuzzy clustering to community ecologists [21]. The fuzzy grouping transformation is a technique used to perform data cleaning tasks while eliminating duplicate data [22] and it is part of *Microsoft* SQL *Server* [23].

Table 6. ANOVA results.

Origin	Type III sum of squares	*df*	Quadratic mean	*F*	Sig.
Corrected model	40.534.397.077	11	3.684.945.189	8.927	0,000
Volume	31.168.599.521	1	31.168.599.521	75.506	0,000
Volume*query type	861.253.945	2	430.626.972	1.043	0,359
Número* query type	2.321.883.863	2	1.160.941.931	2.812	0,068
Error	24.767.737.862	60	412.795.631		
Total	142.395.670.474	72			
Corrected total	65.302.134.939	71			

Related work

Fuzzy groups are one of the fields of mathematics using fuzzy set theory, presented by Rosenfeld [16]. It has also been utilized for several domains of application such as: classification, pattern recognition, image processing, artificial intelligence, information systems, data analysis, decision making and database clustering. Zhang and Huang [17] proposed some SQL instructions to allow grouping in the context of spatial data. Basically, these instructions act as wrappers of conventional grouping algorithms, but no further integration with databases is studied. Li [18] extended the operator GROUP BY to group all tuples approximately within a number of predefined groups. This framework makes use of conventional grouping algorithms, e.g. K-means, and utilizes bitmap indexes to integrate grouping and classification into databases. Silva *et al.* [19] proposed a Group-By based on a similarity principle in PostgreSQL. Laverde [20] introduced a new operator capable of producing higher quality groups for several data domains. In this work, we focus on fuzzy grouping based on vague concepts instead of grouping based on similarity. We also do not rely on discovering groups because the groups are explicitly specified in the query by means of fuzzy partitions. Additionally, the authors of these works did not consider an extension of fuzzy queries such as SQLf, but only extension of a

Conclusions

In this work, the PostgreSQL DBMS was extended to execute fuzzy grouping queries within a tight coupled architecture. To the best of our knowledge, it is the first implementation of this kind. For queries with fuzzy grouping, the catalog was extended to add the new aggregation functions for Fuzzy Grouping as a part of standard PostgreSQL functions. The new functions added were *count_p* and *count_prel*. The proposed extension involved the modification of several modules of the database manager, they were the parser, the planner and the executor. It should be noted that this method can be applied to other extensions that want to be made beyond the fuzzy paradigm. The changes involve the incorporation of new structures and the implementation of new operators to manage these new structures and the operations associated with them. In particular, to evaluate the Fuzzy Group-By (FGB) operator, we proposed a new hash-based algorithm. The algorithm implemented the process of filling a hash table for fuzzy grouping. Also, as new keywords were added to the syntax and new nodes were added to the Parser tree, a new way of grouping tuples using partition tags (classic or fuzzy) was introduced.

This work focused on fuzzy grouping based on vague concepts, rather than similarity-focused grouping.

It did not focus on the discovery of groups since they are explicitly specified in the query, through fuzzy partitions. The experimental study shows that the proposed clustering mechanism, integrated into a database engine, is more efficient than other solutions. The load lies in the use of the implemented aggregation functions, specifically the case of the *count_prel* function, in which it is necessary to execute additional queries for cardinality calculations. Therefore, the greater the volume of records that the table has, the more load is added to the execution time.

As future work, two key points are to be considered in order to enable more complete fuzzy aggregation functions: the implementation of the Fuzzy Grouping defined by Bosc and Pivert [8] and the support of the HAVING clause for the fuzzy query evaluation, in addition to implementing the *count-g* aggregation function.

Acknowledgement

We thank professor Ralph Grove, a friend in Norfolk, VA who helped us with the editing of this paper.

References

[1] Bosc P. and Pivert O.: "SQLf: a relational database language for fuzzy querying". IEEE Transactions on Fuzzy Systems, 3(1), (1995)1-17. https://doi.org/10.1109/91.366566.

[2] Bosc P. and Pivert O.: "SQLf Query Functionality on Top of a Regular Relational Database Management System". Studies in Fuzziness and Soft Computing, (2000), 171-190.

[3] George R., Petry F. E., Buckles B. P. and Srikanth R.: "Fuzzy database systems—challenges and opportunities of a new era". Int J of Intelligent Systems, Vol. 11, No. 9, (1996), 649-659.

[4] Pivert O.: "Contribution à l'interrogation flexible de bases de données: expression et évaluation de requêtes floues". (1991). Doctoral dissertation, Université de Rennes 1.

[5] Galindo J., Urrutia A. and Piattini M.: "Representation of Fuzzy Knowledge in Relational Databases". Fuzzy Databases: Modeling, Design and Implementation, (2006), 145-170.

[6] Goncalves M. and Tineo L.: "SQLf3: an extension of SQLf with SQL3 features". In Proceedings of 10th IEEE International Conference on Fuzzy Systems, (2001), 477-480.

[7] Sanchez, H.R., Sarango, D.E. and Cucuri, M.I.: "Evaluación de un sistema de alimentación avícola basado en lógica difusa". Revista Técnica de Ingeniería Universidad del Zulia, Vol. Especial, No. 1, (2020), 3-10.

[8] Bosc P. and Pivert O.: "On a fuzzy group-by clause in SQLf". International Conference on Fuzzy Systems, (2010), 1-6.

[9] Aguilera A., Cadenas J.T. and Tineo L.: "Fuzzy Querying Capability at Core of a RDBMS". In Advances in Data Mining and Database Management, IGI Global. Hershey, 2011, 160-184.

[10] Bosc P. and Galibourg M.: "Indexing principles for a fuzzy database". J. Information Systems, Vol. 14, No. 6, (1989), 493-499.

[11] Pivert O. and Bosc P.: "Fuzzy Group By". In: Fuzzy preference queries to relational databases. World Scientific, (2012), 251–265.

[12] Timarán R.: "Arquitecturas de Integración del Proceso de Descubrimiento de Conocimiento con Sistemas de Gestión de Bases de Datos: un Estado del Arte, Ingeniería y Competitividad", Vol. 3, No. 2, (2001), 45-55.

[13] Smits G., Pivert O. and Girault T.: "ReqFlex: fuzzy queries for everyone". Proc. VLDB Endow., Vol. 6, No. 12, (2013), 1206-1209.

[14] Aguilera A., Cadenas J. and Tineo L.: "Rendimiento de Consultas SQLf en arquitecturas débil y fuertemente acopladas". Revista Multiciencias, Latindex Venezuela, Vol. 11, No 4, (2011), 410-415.

[15] Cadenas. J.: "Una contribución a la interrogación flexible de bases de datos: Optimización y evaluación a nivel físico", (2006), Master Thesis, USB, Caracas, Venezuela.

[16] Rosenfeld A.: "Fuzzy groups". Journal of mathematical analysis and applications, Vol. 35, No. 3, (1971), 512-517.

[17] Zhang C. and Huang Y.: "Cluster By: a new sql extension for spatial data aggregation". In Proceedings of the 15th annual ACM International Symposium on Advances in Geographic Information Systems, (2007), 1-4.

[18] Li C., Wang M., Lim L., Wang H. and Chang K. C.: "Supporting ranking and clustering as generalized order-by and group-by". In Proceedings of the ACM SIGMOD International Conference on Management of data, (2007), 127-138.

[19] Silva Y. N., Aref W. G. and Ali M. H.: "Similarity group-by". In Proceeding of 2009 IEEE 25th International Conference on Data Engineering, (2009), 904-915.

[20] Laverde N. A., Cazzolato M. T., Traina A. J. and Traina C.: "Semantic Similarity Group By Operators for Metric Data". In Similarity Search and Applications (SISAP), (2017), Vol. 10609, 247-261.

[21] Henderson P.A., Seaby R.M.H. and Somes J.R.: "Fuzzy Grouping". Pisces Conservation Ltd., Lymington, Hampshire, UK., Vol. 2, (2014).

[22] Pudło F. and Ząbkowski T.: "Information Quality improvement methods in Management Information Systems". Information Systems in Management II, Wyd. SGGW, (2008), 124-133.

[23] Zhang J., Guyer C., Milener G. and Petersen T.: "Fuzzy Grouping Transformation". Available at https://docs.microsoft.com/en-us/sql/integration-services/data-flow/transformations/fuzzy-grouping-transformation?view=sql-server-2017, (2017).

UNIVERSIDAD
DEL ZULIA

REVISTA TECNICA

OF THE FACULTY OF ENGINEERING
UNIVERSIDAD DEL ZULIA

Vol. 44. N°1, January - April 2021

This Journal was edited and published in digital format on December 2020 ***by Serbiluz Editorial Foundation***

www.luz.edu.ve
www.serbi.luz.edu.ve
www.produccioncientifica.luz.edu.ve