Priority, Weight and Threshold in Fuzzy SQL Systems

Aleksandar Takači

Faculty of Technology, University of Novi Sad
Bulevar Cara Lazara 1
Serbia
stakaci@tehnol.ns.ac.yu

Srdjan Skrbic

Faculty of Sciences, University of Novi Sad
Trg Dositeja Obradovića 3
Serbia
shkrba@uns.ns.ac.yu

Abstract: PFSQL is the query language used for querying fuzzy relational databases. One of the most distinguished features of PFSQL is the possibility to prioritize conditions. Priorities are most often confused with weights. In this paper we compare queries with prioritized conditions with queries with weighed conditions. Since PFCSP systems are the theoretical background for PFSQL and similarly WFCSP are the theoretical background for weighted queries we elaborate these two systems. Queries with thresholds are another feature of PFSQL. When a threshold is attached to a condition only the tuples that satisfy the condition with a higher value then the threshold are displayed in the result. Through examples we compare these features of PFSQL.

Keywords: PFSQL, PFCSP, priority, threshold, weight

1 Introduction

The representation of imprecise, uncertain or inconsistent information is not possible in relational databases, thus they require add-onns to handle these types of information. One possible add-on is to allow the attributes to have values that are fuzzy sets on the attribute domain, which results in fuzzy relational databases (FRDB) [2].

SQL (Structured Query Language) is the most influential commercially marketed database query language. It uses a combination of relational algebra and relational calculus constructs to retrieve desired data from a database. FSQL (Fuzzy Structured Query Language) is SQL that can handle fuzzy attribute values [1]. The main difference between SQL and FSQL is that SQL returns a subset of the database that matches search criteria as the query result. On the other hand, FSQL returns a subset of the database together with value in the unit interval for each data row that marks how much does that particular data row matches search criteria.

When attributes with fuzzy values appear in the query it is transformed into a query that can be handled by SQL. Finally, results obtained from the SQL query are post processed in order to obtain the desired information as explained above.

Priority is implemented within Prioritized fuzzy constraint satisfaction problem (PFCSP). PFCSP is actually a fuzzy constraint satisfaction problem (FCSP) in which the notion of priority is introduced [4]. Perhaps, the key factors in that implementation are priority t-norms. They are introduced in such a way that the smallest value, usually the value with the biggest priority, has the largest impact on the result given by a priority t-norm. It is introduced by an axiomatic framework. More details about PFCSP are given later in the paper. PFCSP is the theoretical background for incorporating priority into FSQL.

PFSQL allows conditions in the WHERE clause of the FSQL query to have a certain priority i.e. importance degree [5]. Priorities are most often confused with weights. We compare weighed FSQL queries with PFSQL queries. Also each condition in the WHERE clause can have a threshold. If the threshold is not satisfied the data row is dropped from query result.

2 FCSP, PFCSP and Threshold

I will first define FCSP as the background of PFCSP and WFCSP.

Definition 1 A fuzzy constraint satisfaction problem (FCSP) is defined as a 3-tuple (X, D, C^f) where:

1 $X = \{x_i \mid i = 1, 2, \cdots, n\}$ is a set of variables.

2 $D = \{d_i \mid i = 1, 2, \cdots, n\}$ is a finite set of domains. Each domain d_i is a finite set containing the possible values for the corresponding variable x_i in X.

3 C^f is a set of fuzzy constraints. That is,

$$C^f = \{R^f \mid \mu_{R_i^f} : (\prod_{x_j \in var(R^f)} d_j) \to [0,1] \tag{1}$$

where $i = \{1, 2, \cdots n\}$.

PFCSP systems are an extension of FCSP and they are introduced axiomatically. Instead of giving the formal definition of axioms we will briefly explain each of them.

The first axiom states that a zero value of the local satisfaction degree of the constraint with the maximum priority implies a zero value of the local satisfaction degree. The second axiom states that, in the case of equal priorities, the PFCSP becomes a FCSP. The third axiom captures the notion of the priority. If one constraint has a larger priority then, the increase of the value on that constraint should result in a bigger increase of the global satisfaction degree than when the value with the smaller priority has the same increase. It captures the concept of priority in a linear sense. For example take two investments where one of them results in a bigger profit (larger priority) then it is expected that it is better to invest in a more profitable investment then in a less profitable one, if the profit increase is linear to the investment sum.

The fourth axiom is the monotonicity property, and finally the fifth is the upper boundary condition.

When the t-norm T_L is used together with the s-norm S_P we obtain the system that satisfies the given axiomatic framework. The global satisfaction degree in this system is calculated using the following formula:

$$\alpha_\rho(v_X) = \bigoplus\{\frac{\rho(R^f)}{\rho_{max}} \Diamond \mu_{R_i^f}(v_{var(R^f)}) \mid R^f \in C^f\}, \tag{2}$$

where $\bigoplus(x,y) = T_L(x,y)$, and $\Diamond(x,y) = S_P(1-x,y)$. We will call this system $T_L - S_P$.

Similarly, we obtain the $\min - \max$ system if we take $\bigoplus(x,y) = T_M(x,y)$, and $\Diamond(x,y) = S_M(1-x,y)$.

Now, describe how a PFCSP works. Priority of every constraint R_f is evaluated by function $\rho : R_f \to [0,\infty)$. The larger the value of ρ is the larger the priority. After the normalization of the priority values which is done by dividing each priority by $\rho_{\max} = \max\{\rho(R_f), R_f \in C_f\}$ every priority obtains a value in the unit interval. Standard implication aggregates priority of each constraint with its

value. This is done in a way that the larger the priority, the more chance it has for the resulting value to stay the same as it was before aggregation. If the priority of constraint is small, then the aggregated value is closer to 1. This leads to greater values for constraints with the smaller priority. It makes sense, since when these aggregated values are again aggregated with a Scour-concave t-norm T, the smaller values have more impact on the result due to properties of Scour-concave t-norms [3, 6]. We have given two concrete PFCSP systems, min-max and $T_L - S_P$ that satisfy the previously given axioms. Now, we will describe how the global satisfaction degree of this system is calculated.

The function ρ represents the priority of each constraint. Operator $\Diamond$ aggregates priority of each constraint with the value of that constraint. These are then aggregated by the operator $\bigoplus$, which results in the satisfaction degree of an evaluation.

Priorities in PFCSP are most confused with the concept of weights. We can define a WFCSP – weighted fuzzy constraint satisfaction problem, where for each constraint C_i we have an assigned weight w_i. The global satisfaction degree for a valuation v_X, $\alpha_W(v_X)$ in WFCSP is calculated by a known formula:

$$\alpha_W(v_X) = T(c_1 * w_1, c_2 * w_2, \ldots, c_n * w_n), \tag{3}$$

where $c_i = \mu_{C_i}(v_X)$ is the local satisfaction degree of a constraint C_i and T is a t-norm. In order to have an adequate comparison between WFCSP and PFCSP we take $T = T_M$ and $T = T_L$. When T_M is used we get the global satisfaction degree $\alpha_W^{T_M}(v_X)$, and analogously when T_L is used we get $\alpha_W^{T_L}(v_X)$.

In FSQL we can assign a threshold (THOLD) to each constraint. We will now point out the difference between threshold and priority in order to avoid any confusion. If there is a THOLD quantifier attached to a condition, FSQL automatically discards the data row which does not satisfy the condition with a given threshold. On the other hand, if the value of the PRIORITY exists, PFSQL calculates the satisfaction degree for each data row regardless of its satisfaction degree as it will be shown in an example in the following section.

3 FRDB and PFSQL

If we allow the attributes in classical RDB to have values that are fuzzy subsets of the attribute domain, the result will be fuzzy relational databases (FRDB).

Our idea is to have the most common fuzzy set types implemented and that the attribute values in FRDB are most often standard fuzzy sets, and only a small percentage of attribute values are generalized fuzzy sets specified by the user, though our model works with general fuzzy sets in every aspect of FRDB - storing, querying, etc [8]. We introduce one more extension of the attribute value, the linguistic label. Linguistic labels are used to represent most common and widely used expressions of a natural language (such as 'tall people', 'small salary' or 'mediocre result'). Linguistic labels are in fact named fuzzy values from the domain. In order to use a linguistic label on some domain, first we must define this label. For instance, we can define the linguistic label 'tall man' as a fuzzy quantity that has an increasing linear membership function from the point (185,0) to the point (200,1). Considering these extensions, we can define a domain of a fuzzy attribute as:

$$D = D_C \cup F_D \cup L_L, \tag{4}$$

where D_C is a classical attribute domain, F_D is a set of all fuzzy subsets of the domain, and L_L is the set of linguistic labels. In our model we allow triangular fuzzy numbers and fuzzy quantities for F_D.

The basic difference between SQL and PFSQL is in the way the database processes records. In a classical relational database, queries are executed so that a tuple is either accepted in the result set, if it fulfills conditions given in a query, or removed from the result set if it does not fulfill the conditions. In other words, every tuple is given a value true (1) or false (0). On the other hand, as the result set PFSQL returns a fuzzy relation on the database. Every tuple considered in the query is given a value from the unit interval. This value is calculated using operators of fuzzy logic. The question is what elements of the classical SQL should be extended. Because variables can have both crisp and fuzzy values, it is necessary to allow comparison between different types of fuzzy values as well as between fuzzy and crisp values. In other words, PFSQL has to be able to calculate expressions like $height = triangle(180, 11, 8, lin)$, regardless of what value of height is in the database – fuzzy or crisp.

In classical SQL it is clear how to assign truth value to every elementary condition. With fuzzy attributes, situation is more complex. At first, we assign truth value from the unit interval to every elementary condition. Only way to do this is to give algorithm that calculates truth value for every possible combination of values in query and values in the database. For instance, if a query contains

condition that compares a fuzzy quantity value with a triangular fuzzy number in the database, we must have algorithm to calculate compatibility of the two fuzzy sets using a similarity relation. After the truth values from unit interval are assigned, they are aggregated using fuzzy logic. We use t-norm in case of operator AND, and its dual t-conorm in case of operator OR. For negation we use strict negation: $N(x) = 1 - x$.

In case of priority statements, mechanisms deduced from PFCSP systems are used to calculate the result [7]. With normalization of the priority values, every priority obtains a value in the unit interval and also one of the priorities has the value 1. Moreover, with standard implication (*S(1-p, v)*, *S* is a s-norm) we aggregate priority of each constraint with its value. This is done in a way that the larger the priority, the more chance it has for the resulting value to stay the same as it was before aggregation. If the priority of constraint is small, then the aggregated value is closer to 1. This leads to greater values for constraints with the smaller priority. It makes sense, since when these aggregated values are again aggregated with either T_M or T_P the smaller values have more impact on the global satisfaction degree.

We now describe processes that allow PFSQL queries to be executed. The basic idea is to first transform PFSQL query in something that classical SQL interpreter understands. Namely, conditions with fuzzy attributes are removed from WHERE clause and those fuzzy attributes are moved up in the SELECT clause. In this way, conditions containing fuzzy constructs are eliminated, so that the database will return all the tuples – ones that fulfill fuzzy conditions as well as the ones that do not. As a result of this transformation, we get a classical SQL query. Then, when this query is executed against the database, results are interpreted using fuzzy mechanisms. These mechanisms assign a value from unit interval to every tuple in the result set.

More precisely, processing the PFSQL query comprises of four phases: query syntax checking, loading the query into memory structure, transformation of the query, and fuzzy interpretation of the results returned by the database. First, the given query is checked for correct syntax using scanner and parser constructed for this task. If the query is correct, the result of syntax analysis done by the parser is a memory structure that represents this query. Next step is the transformation of this structure in already described fashion. We need to check whether an attribute is fuzzy or not. When the fuzzy attributes are identified, conditions in the WHERE clause that they appear in are removed, and the attributes are added to the SELECT clause. Removed conditions are put in another memory structure, because they will be used to interpret the result set. Result is a classical SQL query which can be directly executed against the database. After the query is executed, returned results are further processed. A measure of condition fulfillment is assigned to every tuple in the result set. This measure is a value from the unit interval which is defined by a similarity relation described in the

following section. In this phase we use a memory structure with fuzzy conditions removed from WHERE clause to calculate measures using priority fuzzy logic.

4 Example

Now we will describe the scenario. Suppose we have to pick a soccer player and a basketball player. We evaluate candidates based on their *Height*, *Speed* and *Stamina*. Depending on the sort, each attribute will have a certain priority. We will suppose that for the soccer player *Speed* is the most important, *Stamina* is mildly important and *Height* is the least important. For the basketball player *Height* is the most important, *Stamina* is mildly important and *Speed* is the least important. The min-max and T_L - S_P systems will be used for evaluation.

The results for five athletes are given in the following tables. Table 1 represents evaluations for the soccer player. We assume that the priority of *Speed* constraint is 1, *Stamina* has priority 0.7 and finally *Height* has priority 0.2. Similarly, Table 2 represents evaluations for the basketball player where priority *Speed* constraint is 0.4, *Stamina* has priority 0.6 and finally *Height* has priority 1.

The example for the soccer player can be interpreted as the following PFSQL query.

SELECT *
FROM Athletes
WHERE (*Height*='tall') PR 0.2
AND (*Stamina*='Excellent') PR 0.6
AND (*Speed*='fast') PR 1

The satisfaction degrees for the query are given in Table 1.

no.	*Spd*	*Sta*	*Hei*	T_M	T_L
1	1	0.6	0.2	0.6	0.53
2	0.55	0.65	0.7	0.55	0.31
3	0.1	0.6	1	0.1	0
4	0.8	0.7	0.7	0.7	0.59
5	0.9	0.6	0.3	0.6	0.59

Table 1
Satisfaction degree for the Soccer player

Similarly, the example for the basketball player can be interpreted as the following PFSQL query.

SELECT *
FROM Athletes
WHERE (*Height*='tall') PR 1
AND (*Stamina*='Excellent') PR 0.6
AND (*Speed*='fast') PR 0.4
The satisfaction degrees for the query are given in Table 2.

no.	*Spd*	*Sta*	*Hei*	T_M	T_L
1	1	0.6	0.2	0.7	0.575
2	0.55	0.65	0.7	0.25	0.145
3	0.1	0.6	1	0.1	0.07
4	0.8	0.7	0.7	0.7	0.5
5	0.9	0.6	0.3	0.6	0.51

Table 2
Satisfaction degree for the Basketball player

If we want to use a WFCSP - T_M query for choosing the athletes, the query for the soccer player would be the following.

SELECT (MIN(Height*0.2,Stamina*0.6,Speed*1))
FROM Athletes
WHERE (*Height*='tall')
AND (*Stamina*='Excellent')
AND (*Speed*='fast')

Similarly, for WPFCSP - T_L the query should have the following form:

SELECT (MAX(Height*0.2+Stamina*0.6+Speed*1,0))
FROM Athletes
WHERE (*Height*='tall')
AND (*Stamina*='Excellent')
AND (*Speed*='fast')

The satisfaction degrees for the queries are given in Table 3.

no.	*Spd*	*Sta*	*Hei*	T_M	T_L
1	1	0.6	0.2	0.2	0
2	0.55	0.65	0.7	0.11	0
3	0.1	0.6	1	0.002	0
4	0.8	0.7	0.7	0.16	0
5	0.9	0.6	0.3	0.18	0

Table 3
Weighted satisfaction degrees for the Soccer player

We see that the results in Table 3 differ completely from Table 1. Moreover, if we would run the weighed query for the basketball player we would obtain similar results. This leads to a conclusion that weighted queries are completely different from priority queries.

Finally we can add a threshold to each of the constraints. Assume that we insist that a basketball player must be tall, has good stamina with the degree of 0.6 and fast with the degree of 0.4. This would translate into the following query.

SELECT *
FROM Athletes
WHERE (*Height*='tall') THRESHOLD 1
AND (*Stamina*='Excellent') THRESHOLD 0.6
AND (*Speed*='fast') THRESHOLD 0.4

It is obvious that none of the athletes would fulfill these requirements. This situation is completely different than when we used priority or weighted queries.

Conclusions

In this paper we have presented the PFSQL language and PFCSP systems as the theoretical background for PFSQL, as well as some directions about PFSQL implementation. Similarly we have given a brief description of WFCSP as the theoretical background for weighted queries. In addition, a description of queries with thresholds is also given as another feature of PFSQL.

We have discussed differences and given a comparison of priority, weighted and threshold queries using an example of choosing athletes. We have shown that these three types of queries make different evaluations for the same data i.e. they are essentially different. The threshold is the most strict decision mechanism. Weighted and priority queries are based on different assumptions and must not be confused.

Acknowledgment

The authors would like to acknowledge the support of the Serbian Ministry of Science and Environmental Protection, project 'Mathematical models of non-linearity, uncertainty and decision making', No. 144012 and project 'Abstract Methods and Applications in Computer Science' No. 144017A, also the support of the Ministry of Science, Technology and Environmental Protection of Vojvodina.

References

[1] Galindo, J., Urrutia, A., Piattini, M.: Fuzzy Databases: Modeling Design and Implementation. Hershey, USA: IDEA Group, 2006

[2] Kerre, E. E., Chen, G. Q.: Fuzzy Data Modeling at a Conceptual Level: Extending ER/EER Concepts, In Knowledge Management in Fuzzy Databases, 2000, pp. 3-11

[3] Klement, E., Mesiar, R., Pap, E.: Triangular Norms, Series: Trends in Logic (8), Dordrecht: Kluwer Academic Publishers, 2000

[4] Leung, H., Jennings, N. R.: Prioritized Fuzzy Constraint Satisfaction Problems: Axioms, Instantiation and Validation, Fuzzy Sets and Systems 136(10), 2003, pp. 151-188

[5] Takači, A., Škrbić, S.: How to Implement FSQL and Priority Queries. Proceedings of 3rd Serbian-Hungarian Joint Symposium on Intelligent Systems, Subotica, Serbia: Budapest Tech Polytechnical Institution, 2005, pp. 98-104

[6] Takači, A.: Schur-Concave Triangular Norms: Characterization and Application in PFCSP, Fuzzy Sets and Systems, 155(1), 2005, pp. 50-64

[7] Takači, A., Škrbić, S.: Data Model of FRDB with Different Data Types and PFSQL, Handbook of Research on Fuzzy Information Processing in Databases, Hershey, PA, USA: Information Science Reference, in print, 2008

[8] Zvieli, A., Chen, P.: ER Modeling and Fuzzy Databases, Proceedings of the Second International Conference on Data Engineering, Los Angeles: IEEE Computer Society, 1986, pp. 320-327