Bipolar SQLf: a Flexible Querying Language for Relational Databases

Nouredine Tamani, Ludovic Lietard, Daniel Rocacher

Bipolar SQLf: a Flexible Querying Language for Relational Databases

Nouredine Tamani[1], Ludovic Liétard[2], and Daniel Rocacher[1]

[1] IRISA/ENSSAT/University Rennes 1,
6 rue de Kerampont, 22300 Lannion, France
{tamani,rocacher}@enssat.fr
[2] IRISA/IUT/University Rennes 1,
Rue Edouard Branly, BP 30219 22302 Lannion Cedex, France
ludovic.lietard@univ-rennes1.fr

Abstract. Flexible querying of information systems allows expressing complex preferences in user queries. Such preferences can be modeled by fuzzy bipolar conditions which are made of constraints c and wishes w and interpreted as *"to satisfy c and if possible to satisfy w"*.We define in this article the main elements of the Bipolar SQLf language, which is an SQL-like querying language based on a bipolar relational algebra [11, 3]. This language is an extension of the SQLf language [2, 1]. Basic statements (projection, selection, etc.) are firstly defined in terms of syntax, evaluation and calibration. Then, complex statements, such as bipolar queries based on nesting operators are studied in terms of expression, evaluation, query equivalence and backward compatibility with the SQLf language.

Keywords: Flexible Querying, Fuzzy Bipolar Conditions, Fuzzy Bipolar Algebra, SQLf Language, Bipolar SQLf Language.

1 Introduction

Flexible querying of databases allows users to express preferences in their queries. Within this framework, numerous tools have been proposed such as Preference SQL [5], SQLf language [2, 1], Top-k queries [9], the winnow operator [8], etc. which are based on diverse mathematic foundations.

In the context of fuzzy querying, user preferences are expressed by fuzzy predicates (such as *high*, *fast*, *expensive*, etc.) which are defined by fuzzy sets [13]. The SQLf language is an extension of the SQL language to fuzzy conditions, which allows expressing queries addressed to relational databases. Such queries deliver a set of tuples attached with degrees used to rank them from the most to the least preferred. In this context, it is also possible to consider fuzzy bipolar conditions to model preferences.

A bipolar condition is a compound condition made of negative and positive conditions. Several interpretations have been introduced for the evaluation of queries involving such conditions (see [6, 7, 12, 14, 15]). In this paper, we rely

on the interpretation introduced by Dubois and Prade [6, 7], in which a bipolar condition is made of a constraint c and a wish w. It is noted (c, w) and means *"to satisfy c and if possible to satisfy w"*. More precisely, for the expression of user preferences, we rely on fuzzy bipolar conditions in which the constraint and the wish are defined by fuzzy sets. Furthermore, we define a fuzzy bipolar query as a query that involves fuzzy bipolar conditions. More precisely, when querying a relation R with a fuzzy bipolar condition, each tuple t from R is then attached with a couple of grades $(\mu_c(t), \mu_w(t))$ that expresses the degree of its satisfaction to c and w and a so-called fuzzy bipolar relation is obtained.

In order to define a bipolar relational algebra, the algebraic operators (selection, projection, join, union, intersection) have been extended to fuzzy bipolar conditions [11, 3]. These operators allow the expression of fuzzy bipolar queries.

We are aimed in this article to define the Bipolar SQLf language which is an SQL-like language based on a bipolar relational algebra. Since fuzzy bipolar conditions generalize fuzzy conditions, we consider the enrichment to fuzzy bipolar conditions of the SQLf language [2, 1] which is devoted to flexible querying with fuzzy sets. At the first step basic Bipolar SQLf queries are defined in terms of expression, evaluation and calibration. Then, complex bipolar queries based on nesting ($in_{=}$, $in_{\approx}$, $exists$, θany) and partitioning operators are defined.

The remainder of this paper is as follows. In section 2, both fuzzy sets theory and the SQLf language are described. Section 3 introduces respectively fuzzy bipolar conditions, the bipolar relational algebra defined in [11, 3] and the basis of the Bipolar SQLf language. In section 4, the extension of advanced Bipolar SQLf statements to fuzzy bipolar conditions is studied. Section 5 sums up our contribution and draws some lines for future works.

2 Flexible Querying Within the SQLf Language

We introduce in this section the fuzzy sets theory and the SQLf language.

2.1 The Fuzzy Sets Theory

The fuzzy sets theory is introduced by Zadeh [13] to express the gradual membership of an element to a set. Formally, a fuzzy set F is defined on a referential U by a membership function $\mu_F : U \mapsto [0, 1]$ such that $\mu_F(x)$ denotes the membership grade of x in F. In particular, $\mu_F(x) = 1$ denotes the full membership of x in F, $\mu_F(x) = 0$ expresses the absolute non-membership and when $0 < \mu_F(x) < 1$, it reflects a partial membership (the closer to 1 $\mu_F(x)$, the more x belongs to F).

A fuzzy set generalizes an ordinary (crisp) set in which membership grades are in $\{0, 1\}$. If a fuzzy set is a discrete set then it is denoted $F = \{\mu_F(x_1)/x_1, ..., \mu_F(x_n)/x_n\}$, otherwise, it is characterized by its membership function, generally a trapezoidal function.

The union $\cup$ and the intersection $\cap$ operators are defined with a couple of (t-norm, t-conorm) such as (min, max). Let F, G be two fuzzy sets, $\mu_{F \cup G}(x) =$

$max(\mu_F(x), \mu_G(x))$, $\mu_{F\cap G}(x) = min(\mu_F(x), \mu_G(x))$, and the complement of F, noted F^c, is defined by $\mu_{F^c}(x) = 1 - \mu_F(x)$.

The logical counterparts of $\cap$, $\cup$ and the complement are resp. $\wedge, \vee$ and $\neg$. Other operators have also been defined such as fuzzy implications [4].

2.2 The SQLf Language

The SQLf language [2, 1] is an extension of the SQL language to flexible querying based on fuzzy conditions. An SQLf query delivers a fuzzy relation r where the grade of membership of tuple t expresses its level of satisfaction.

The SQLf language is based on the fuzzy relational algebra, in which relational algebra operators have been extended to fuzzy predicates as follows: let r, s be two fuzzy relations such that the schema of r (resp. s) is X (resp. Y):
Fuzzy projection: $\mu_{\pi(r,V)}(v) = max_w \mu_r(vw)$, where $V \subseteq X$, $v \in V$ and $w \in (X - V)$.
Fuzzy selection: $\mu_{\sigma(r,p)}(x) = min(\mu_r(x), \mu_p(x))$, where p is a fuzzy predicate.
Fuzzy join: $\mu_{\bowtie}(r, s, \theta, A, B) = min(\mu_r(x), \mu_s(y), \mu_\theta(x.A, y.B))$, where A and B are compatible sets of attributes such that A (resp. B) is a subset of X (resp. Y) and $x.A$ (resp. $y.B$) is the value of A in x (resp. B in y), and θ is either a crisp or a fuzzy binary operator ($\theta \in \{=, \approx, <, >, \textit{much larger than}, ...\}$).

The basic form of an SQLf query is a fuzzy restriction defined as follows:

***Select** [**distinct**] [n|t|n,t] attributes **From** relations **Where** fuzzy_cond;*

This query returns a set of ranked tuples with their attached degree, where n specifies an n-top query and $t \in]0, 1]$ is a minimal threshold of satisfaction.

Example 1: Let *fast* be a fuzzy predicate defined on $\mathbb{R}_+ \to [0, 1]$: $\mu_{fast}(d) = 1$, if $d \in [0, 2]$; $\frac{-d}{3} + \frac{5}{3}$, if $d \in [2, 5]$ and 0, *otherwise*. The query *"find the 2 fastest journeys from Brussels to Paris"* can be expressed in SQLf by:

Select** 2 #journey **From** Journey **Where
*source='Brussels' **and** destination='Paris' **and** fast (duration);*

The fuzzy condition *fast* delivers the fuzzy relation *FastJourney*, where $\mu_{FastJourney}(t) = \mu_{Fast}(t.duration)$. Table 1 is an example of the fuzzy relation *fastJourney* and journey #12 and either journey #13 or #10 are delivered. ■

Table 1. Extension of the fuzzy relation *fastJourney*.

#Journey	cost ($)	duration (h)	...	$\mu_{FastJourney}$
12	70	2	...	1
13	50	3	...	0.66
10	50	4	...	0.66

The SQLf language allows the expression of more complex statements such as partitioning, nesting and division involving fuzzy relations. For example, the query *"find journeys for which most of their steps are comfortable"* corresponds to a partitioning based on fuzzy quantified propositions.

3 Flexible Querying and Bipolarity

In this section, we introduce fuzzy bipolar conditions, a bipolar relational algebra [11, 3] and the basis of Bipolar SQLf language.

3.1 Fuzzy Bipolar Conditions

A bipolar condition is a compound condition which is made of two conditions defined on the same universe: i) a constraint c, which describes the set of acceptable elements, and ii) a wish w which defines the set of desired elements. Since it is incoherent to wish a rejected element, the property of coherence $w \subseteq c$ holds.

It is worth mentioning that the linguistic expression of a fuzzy condition may not follows the coherence property. As an example, a user may think of *"a japanese car and if possible a red car"*, however, such a condition should be clearly be rewritten *"a japanese car, and if possible a japanese and red car"*.

In addition, condition c is mandatory in the sense that an element which does not satisfy it is rejected. Condition w is optional in the sense that an element which does not satisfy it is not necessarily rejected.

If c and w are boolean conditions, the satisfaction with respect to (c, w) is a couple from $\{0, 1\}^2$. When querying a database with such a condition, tuples satisfying the constraint and the wish are returned in priority to the user. If such answers do not exist, tuples satisfying only the constraint are delivered.

If c and w are fuzzy conditions (defined on the universe U), the property of coherence becomes: $\forall u \in U, \mu_w(u) \leq \mu_c(u)$ and the satisfaction with respect to (c, w) is a couple of degrees from $[0, 1] \times [0, 1]$. Each element u from U is then attached with a pair of grades $(\mu_c(u), \mu_w(u))$ that expresses the degree of its satisfaction respectively to the constraint c and to the wish w.

In the context of bipolar relations, a tuple t is then denoted $(\mu_c, \mu_w)/t$. We assume that any tuple u such that $\mu_c(u) = 0$ does not belong to the fuzzy bipolar relation. In addition, tuples cannot be ranked from the most to the least preferred using an aggregation of μ_c and μ_w because constraints and wishes are not commensurable. However they can be ranked using the lexicographical order: t_1 is preferred to t_2, denoted $t_1 > t_2$ or $(\mu_c(t_1), \mu_w(t_1)) > (\mu_c(t_2), \mu_w(t_2))$, iif $\mu_c(t_1) > \mu_c(t_2)$ or $(\mu_c(t_1) = \mu_c(t_2) \wedge \mu_w(t_1) > \mu_w(t_2))$. Since the constraint is mandatory, its satisfaction is firstly used to discriminate among answers. The satisfaction with respect to the wish being not mandatory, it can only be used to discriminate among answers having the same evaluation with respect to the constraint. A total order is then obtained on c and w (with $(1, 1)$ as the greatest element and $(0, 0)$ as the least element).

Based on the lexicographical order, the *lmin* and *lmax* operators [10, 3] are used to define the conjunction (resp. intersection) and the disjunction (resp. union) of bipolar conditions (resp. relations).

They are defined on $([0, 1] \times [0, 1])^2 \rightarrow [0, 1] \times [0, 1]$ as follows:

$$((\mu, \eta), (\mu', \eta')) \mapsto lmin((\mu, \eta), (\mu', \eta')) = \begin{cases} (\mu, \eta) & \text{if } \mu < \mu' \vee (\mu = \mu' \wedge \eta < \eta'), \\ (\mu', \eta') & \text{otherwise.} \end{cases}$$

$$((\mu,\eta),(\mu',\eta')) \mapsto lmax((\mu,\eta),(\mu',\eta')) = \begin{cases}(\mu,\eta) & \textit{if } \mu > \mu' \vee (\mu = \mu' \wedge \eta > \eta'),\\ (\mu',\eta') & \textit{otherwise.}\end{cases}$$

The $lmin$ (resp. $lmax$) operator is commutative, associative, idempotent and monotonic. The couple of grades $(1,1)$ is the neutral (resp. absorbing) element of the operator $lmin$ (resp. $lmax$) and the couple $(0,0)$ is the absorbing (resp. neutral) element of the operator $lmin$ (resp. $lmax$).

It can be noticed that a fuzzy predicate is a fuzzy bipolar predicate such that $\forall x, (\mu_c(x) = \mu_w(x))$, which means that a fuzzy condition is a fuzzy bipolar condition in which the wish is equal to the constraint. In other words, fuzzy bipolar conditions generalize fuzzy condition and it has been proven [10, 3] that $lmin$ (resp. $lmax$) generalizes the t-norm min (resp. the t-conorm max).

3.2 Basis of the Bipolar Relational Algebra

We introduce the bipolar relational algebra proposed in [11, 3]. It is based on the couple $(lmin, lmax)$. Let r and s be two fuzzy bipolar relations defined respectively by the fuzzy bipolar conditions (c_1, w_1) and (c_2, w_2).

The intersection: $r \cap s$ is a fuzzy bipolar relation defined as follows:

$$r \cap s = \{(\mu_c,\mu_w)/t | (\mu_{c_1},\mu_{w_1})/t \in r \wedge (\mu_{c_2},\mu_{w_2})/t \in s \wedge (\mu_c,\mu_w) = lmin((\mu_{c_1}(t),\mu_{w_1}(t)),(\mu_{c_2}(t),\mu_{w_2}(t)))\}.$$

The union: $r \cup s$ is a fuzzy bipolar relation defined as follows:

$$r \cup s = \{(\mu_c,\mu_w)/t | (\mu_{c_1},\mu_{w_1})/t \in r \wedge (\mu_{c_2},\mu_{w_2})/t \in s \wedge (\mu_c,\mu_w) = lmax((\mu_{c_1}(t),\mu_{w_1}(t)),(\mu_{c_2}(t),\mu_{w_2}(t)))\}.$$

The cartesian product: $r \otimes s$ is a fuzzy bipolar relation defined as follows:

$$r \otimes s = \{(\mu_c,\mu_w)/t \oplus t' | (\mu_{c_1},\mu_{w_1})/t \in r \wedge (\mu_{c_2},\mu_{w_2})/t' \in s \wedge (\mu_c,\mu_w) = lmin((\mu_{c_1}(t),\mu_{w_1}(t)),(\mu_{c_2}(t'),\mu_{w_2}(t')))\},$$

where $\oplus$ is the operator of concatenation of tuples.

The projection π: the projection of distinct tuples of r on attributes $a_1, ..., a_k$ is a fuzzy bipolar relation of tuples $\langle a_1, ..., a_k\rangle$ defined by:

$$\pi_{a_1,...,a_k}(r) = \{(\mu'_{c_1},\mu'_{w_1})/\langle a_1,...,a_k\rangle \mid (\mu'_{c_1},\mu'_{w_1}) = lmax_{t\in r \wedge t[a_1,...,a_k]=\langle a_1,...,a_k\rangle}((\mu_{c_1}(t),\mu_{w_1}(t)))\},$$

where $t[a_1, ..., a_k]$ is the value of the tuple t on the attributes $a_1, ..., a_k$.

The selection σ: the selection of tuples from r, based on the fuzzy bipolar condition (c', w') is defined as:

$$\sigma(r,(c',w')) = \{(\mu_c,\mu_w)/t | (\mu_{c_1},\mu_{w_1})/t \in r \wedge (\mu_c,\mu_w) = lmin((\mu_{c'}(t),\mu_{w'}(t)),(\mu_{c_1}(t),\mu_{w_1}(t)))\}.$$

The join operator $\bowtie$: as in the SQL and SQLf languages, the join operator is defined by the bipolar selection operator applied over a bipolar cartesian product. This operator is studied in Section 4.

3.3 Bipolar SQLf Basic Statements

A Bipolar SQLf basic statement is a fuzzy bipolar selection applied over a bipolar projection operator. It has the following form:

***Select** [**distinct**] [$n|t|(t_1,t_2)|n,t|n,(t_1,t_2)$] attributes **From** relations [**as** alias] **Where** (c,w);*

The parameters intended to calibration of the result are also extended to bipolarity. A bipolar top k query is obtained by positioning the optional integer n, which delivers the n best tuples attached with the n greatest couples of degrees in the sense of the lexicographical order. The qualitative calibration can be specified by either a threshold $t \in [0,1]$, which delivers tuples u such that $(\mu_c(u), \mu_w(u)) \geq (t,0)$, or a couple of thresholds $(t_1,t_2) \in [0,1]^2$, such that $t_2 \leq t_1$, which delivers tuples u such that $(\mu_c(u), \mu_w(u)) \geq (t_1,t_2)$.

Example 2: The query *"find the 2 best journeys which are fast **and if possible** not expensive"*, can be expressed in Bipolar SQLf as:

***Select** 2 #Journey **From** Journey **as** J **Where** $(fast(J.duration), notexpensive(J.cost))$;*

Due to the coherence property of fuzzy bipolar conditions, the fuzzy bipolar condition *"fast **and if possible** and not expensive"* is interpreted as *"fast **and if possible** (fast and not expensive)"*.

The fuzzy predicate *expensive* can be defined as $\forall x \in \mathbb{R}_+$: $\mu_{Expensive}(x) = \frac{x}{80}$, *if* $x \in [0,80]$; 1, *otherwise*; where x expresses the cost of a journey. Its negation is defined as follows: $\forall x \in \mathbb{R}_+, \mu_{notExpensive}(x) = 1 - \mu_{Expensive}(x)$.

Based on the definition of fuzzy predicates *fast* (example 1) and *not expensive*, the query is evaluated over the relation *Journey* and delivers the fuzzy bipolar relation $Journey_{(Fast,\ notExpensive)}$ (see Table 2). The returned tuples are ranked using the lexicographical order : $(1, 0.13)/12$, $(0.66, 0.38)/13$. The tuple #12 is the best one with regard to the constraint (total satisfaction), and tuples #13 and #10 have the same satisfaction with respect to the constraint but #13 is better than #10 on the wish. ■

Table 2. Fuzzy bipolar relation $Journey_{(Fast,\ notExpensive)}$.

#Journey	cost ($)	duration (h)	μ_{Fast}	$\mu_{Fast} \wedge \mu_{notExpensive}$
12	70	2	1	0.13
13	50	3	0.66	0.38
10	50	4	0.66	0.33

4 Extension of Complex SQLf Statements to Bipolarity

In this section, we define the bipolar join operator. Then, the extension to bipolarity of nesting operators ($in_=$, $in_\approx$, $exists$, θany) and aggregate functions are defined in the scope of their equivalence to the bipolar join operator. Since fuzzy bipolar conditions generalize fuzzy conditions, the bipolar definition of these operators is based on the extension to bipolarity of the SQLf statements.

4.1 Extension of the Join Operator to Bipolarity

The basic form of an SQLf join query is as follows:
Q_1: *Select R.A, R′.B From R, R′ Where* $c_1(R)$ *and* $c_2(R')$ *and* $R.att_1 = R'.att_2$;
where c_1 and c_2 are two fuzzy conditions applied resp. on relations R and R'.

Tuples $u = (a, b)$ delivered from Q_1 are attached with degrees processed by the following formula (1):

$$\mu_{Q_1}(u) = \max_{t \in R \wedge t' \in R' \wedge t.A = a \wedge t'.B = b} min(\mu_R(t), \mu_{c_1}(t), \mu_{R'}(t'), \mu_{c_2}(t'), \mu_{=}(t.att_1, t'.att_2)) \quad (1)$$

The formula (1) can be rewritten as follows:

$$\mu_{Q_1}(u) = \max_{t \in R \wedge t' \in R' \wedge t.A = a \wedge t'.B = b \wedge t.att_1 = t'.att_2} min(\mu_R(t), \mu_{c_1}(t), \mu_{R'}(t'), \mu_{c_2}(t')) \quad (2)$$

In the context of bipolarity, the basic form of a join query is as follows:
Q_2: *Select R.A, R′.B From R, R′ Where*
$(c_1(R), w_1(R))$ *and* $(c_2(R'), w_2(R'))$ *and* $R.att_1 = R'.att_2$;
The definition of the join operator based on the formula (1) is as follows:

$$(\mu_{cQ_2}(u), \mu_{wQ_2}(u)) = \underset{t \in R \wedge t' \in R' \wedge t.A = a \wedge t'.B = b}{lmax} lmin((\mu_{R_c}(t), \mu_{R_w}(t)), (\mu_{c_1}(t), \mu_{w_1}(t)), (\mu_{R'_c}(t'), \mu_{R'_w}(t')), (\mu_{c_2}(t'), \mu_{w_2}(t')), (\mu_{=}(t'.att_2, t.att_1), \mu_{=}(t'.att_2, t.att_1))) \quad (3)$$

Based on the formula (2), tuples $u = (a, b)$ delivered from Q_2 are attached with couples of degrees processed by the following formula (4):

$$(\mu_{cQ_2}(u), \mu_{wQ_2}(u)) = \underset{t \in R \wedge t' \in R' \wedge t.A = a \wedge t'.B = b \wedge t.att_1 = t'.att_2}{lmax} lmin((\mu_{R_c}(t), \mu_{R_w}(t)), (\mu_{c_1}(t), \mu_{w_1}(t)), (\mu_{R'_c}(t'), \mu_{R'_w}(t')), (\mu_{c_2}(t'), \mu_{w_2}(t'))) \quad (4)$$

It is easy to prove that formulas (4) and (3) are equivalent.

Remark: A bipolar θ-join operator, where θ is either a boolean or a fuzzy relational operator ($\theta \in \{<, >, \leq, \geq, =, \neq, around, much\ greater\ than, ...\}$), can straightforwardly be defined from formula (3) by substituting $(\mu_{=}(t'.att_2, t.att_1), \mu_{=}(t'.att_2, t.att_1))$ by $(\mu_\theta(t'.att_2, t.att_1), \mu_\theta(t'.att_2, t.att_1))$.

4.2 Bipolar (θ_c, θ_w)-join operator

We define a new bipolar join operator denoted (θ_c, θ_w)-join made of two relational operators: θ_c and θ_w which are in $\{<, >, \leq, \geq, =, \neq, around, greater\ than, ...\}$.This bipolar operator permits us to express queries such as *"find salespersons who get a turnover much greater* ***and if possible*** *very much greater than 10 times their*

own salary". The main form of such queries is:
Q'_2*: Select R.A, R'.B From R, R' Where* $(c_1(R), w_1(R))$ *and*
$(c_2(R'), w_2(R'))$ *and* $(R.att_1\ \theta_c\ R'.att_2, R.att_3\ \theta_w\ R'.att_4)$*;*

Based on the formula (3), couples of degrees associated to tuples delivered from Q'_2 are processed by the following formula (5):

$$(\mu_{cQ'_2}(u), \mu_{wQ'_2}(u)) = \underset{t\in R\wedge t'\in R'\wedge t.A=a\wedge t'.B=b}{lmax} lmin((\mu_{R_c}(t), \mu_{R_w}(t)), (\mu_{c_1}(t), \mu_{w_1}(t)), (\mu_{R'_c}(t'), \mu_{R'_w}(t')), (\mu_{c_2}(t'), \mu_{w_2}(t')), (\mu_{\theta_c}(t'.att_2, t.att_1), \mu_{\theta_w}(t'.att_3, t.att_4))) \quad (5)$$

Example 3: In order to select the best young sellers, a manager based on the monthly balance sheets can express the following query *"find young* ***and if possible*** *very young salespersons with turnovers of their low* ***and if possible*** *very low monthly balance sheets, are much greater* ***and if possible*** *very much greater than* 5 *times their salary"*. It can be written in Bipolar SQLf as:
Select #Seller From Seller as S, MonthBalance as MB Where
S.#Seller = MB.#Seller and (young (S.age), very young (S.age)) and
*(low (MB.turnover), very low (MB.turnover)) and (much greater (S.salary*5, MB.turnover), very much greater (S.salary*5, MB.turnover));*

Due to the space limitation, we only describe the derived fuzzy bipolar relations: $Seller_{(Young,veryYoung)}$ (see Table 3) and $Balance_{(low,veryLow)}$, in which we show the couples of degrees of satisfaction to the bipolar join condition *(much greater(S.salary*5,MB.turnover),very much greater(S.salary*5, MB.turnover)))*.

The predicate *low* is defined on $\mathbb{R}_+ \to [0,1]$ as: $\mu_{low}(x) = 1$ if $x \in [0, 25000]$, $\mu_{low}(x) = \frac{-x}{5000} + 6$ if $x \in [25000, 30000]$, $\mu_{low}(x) = 0$ *otherwise.*

The predicate *very low* is defined as $\forall x \in \mathbb{R}_+ : \mu_{veryLow}(x) = (\mu_{low}(x))^2$.

We define the predicate *much greater* on $(\mathbb{R}^2_+ \to [0,1])$: $\mu_{muchGreater}(x,y) = 1 - \frac{y}{x}$ if $x > y$; 0, otherwise; and the predicate *very much greater* is defined: $\forall (x,y) \in \mathbb{R}^2_+ : \mu_{vMGreater}(x,y) = (\mu_{mGreater}(x,y))^2$.

Table 3. The fuzzy bipolar relation $Seller_{(young,veryYoung)}$.

$\#Seller$	salary	μ_{young}	$\mu_{veryYoung}$
5	2000	1	1
1	2500	0.8	0.64
3	2800	0.2	0.04

The couple of degrees associated to the seller #1 is:
$(\mu_c(\#1), \mu_w(\#1)) = lmax(lmin((0.8, 0.64), (0.5, 0.25), (0.55, 0.33)),$
$lmin((0.8, 0.64), (0.7, 0.49), (0.53, 0.28))) = lmax((0.5, 0.25), (0.53, 0.28)) = (0.53, 0.28)$.

For sellers #3 and #5, the couples of degrees are respectively:
$(\mu_c(\#3), \mu_w(\#3)) = (0.2, 0.04)$ and $(\mu_c(\#5), \mu_w(\#5)) = (0.6, 0.36)$.
Sellers are delivered as follows: $(0.6, 0.36)/\#5, (0.53, 0.28)/\#1, (0.2, 0.04)/\#3$. ■

Table 4. The fuzzy bipolar relation $Balance_{(low,veryLow)}$.

$\#Balance$	$\#Seller$	turnover	μ_{low}	$\mu_{veryLow}$	$\mu_{mGreater}$	$\mu_{vMGreater}$
1	1	27500	0.5	0.25	0.55	0.30
2	1	26500	0.7	0.49	0.53	0.28
5	3	28500	0.3	0.09	0.56	0.32
6	3	28000	0.4	0.16	0.55	0.31
9	5	29500	0.1	0.01	0.66	0.44
10	5	25000	1	1	0.60	0.36

4.3 Extension of $in_{=}$ and $in_{\approx}$ Operators to Bipolarity

In the SQLf language, the $in_{=}$ (resp. $in_{\approx}$) operator expresses at what level a value of an attribute is equal (resp. is close) to a value from the fuzzy set delivered by the nested subquery. The main format of an in_{θ} query, where $\theta \in \{=, \approx\}$, in the SQLf language is:

Q_3*: Select A From R Where* c_1 *and* att_1 in_{θ} *(Select* att_2 *From* R' *Where* c_2*);*

The query Q_3 is equivalent to the following join query [2]:

Q_4*: Select R.A From* R, R' *Where* $R.att_1 \theta R'.att_2$ *and* $c_1(R)$ *and* $c_2(R')$*;*

Based on this equivalence, the evaluation of the condition v_1 such that: $v_1 = att_1$ in_{θ} *(Select* att_2 *From* R' *Where* c_2*)* is as follows:

$$\mu_{v_1} = \max_{t' \in R'} min(\mu_{R'}(t'), \mu_{c_2}(t'), \mu_{\theta}(t'.att_2, t.att_1)) \quad (6)$$

This equivalence holds in the case of bipolarity. The condition v_1 is written $v'_1 = att_1$ in_{θ} *(Select* att_2 *From* R' *Where* (c_2, w_2)*)*.

The evaluation of the condition v'_1 is based on the extension of the formula (6) to bipolarity as follows:

$$(\mu_{c\_v'_1}, \mu_{w\_v'_1}) = \operatorname*{lmax}_{t' \in R'} lmin((\mu_{R'_c}(t'), \mu_{R'_w}(t')), (\mu_{c_2}(t'), \mu_{w_2}(t')), (\mu_{\theta}(t'.att_2, t.att_1), \mu_{\theta}(t'.att_2, t.att_1))) \quad (7)$$

4.4 Bipolar $in_{(\approx,=)}$ Operator

It is possible to define a bipolar *in* operator denoted $in_{(\approx,=)}$ which expresses conditions v_2 of the following form:

$v_2 = att_1$ $in_{(\approx,=)}$ *(Select* att_2 *From* R' *Where* (c_2, w_2)*)*, which expresses at what level att_1 is close ***and if possible*** is equal to a value among those delivered from the bipolar SQLf subquery *(Select* att_2 *From* R' *Where* (c_2, w_2)*)*. From the syntactic point of view the bipolar operator $in_{(\approx,=)}$ is expressed *(approx, equal)*.

The evaluation of the condition v_2 is based on the following formula (8), which is an extension to bipolarity in the formula (6):

$$(\mu_{c\_v_2}, \mu_{w\_v_2}) = \operatorname*{lmax}_{t' \in R'} lmin((\mu_{R'_c}(t'), \mu_{R'_w}(t')), (\mu_{c_2}(t'), \mu_{w_2}(t')), (\mu_{\approx}(t'.att_2, t.att_1), \mu_{=}(t'.att_2, t.att_1))) \quad (8)$$

It is worth noticing that a query defined with the bipolar $in_{(\approx,=)}$ operator is equivalent to a (θ_c, θ_w)-join query where θ_c corresponds to the $\approx$ operator and θ_w corresponds to the $=$ operator.

Example 4: We consider the following query *"find villas which are small,* ***and if possible*** *not far from the downtown and having a price similar,* ***and if possible*** *equal to the price of apartments which are spacious,* ***and if possible*** *located near to the downtown".* It can be expressed in the Bipolar SQLf as follows:
Select #villa From Villa as V Where (small(V.surface), not far_town(V.address)) and V.price (approx, equal) (Select #apart From Apartment as A Where (spacious (A.surface), near_town (A.address))); ■

4.5 Extension of the *exists* Operator to Bipolarity

In the scope of the SQLf language, the *exists* operator indicates a non emptiness of a fuzzy set. It is defined by the formula $\mu_{exists}(E) = sup_{x \in support(E)} \mu_E(x)$ which expresses at what extent an element belongs to the returned fuzzy set.

The main form of an *exists* query in the SQLf language is:

Q_5*: Select A From R Where* c_1 *and exists (Select * From* R' *Where* c_2 *and* $R.att_1 \theta R'.att_2$*);*

where θ is a relational operator which can be either boolean or fuzzy.

This query is equivalent to the following join query:

Q_6*: Select A From* R, R' *Where* $c_1(R)$ *and* $c_2(R')$ *and* $R.att_1$ θ $R'.att_2$*;*

The condition v_3 defined by the *exists* operator:

v_3 = *exists (Select * From* R' *Where* c_2 *and* $R.att_1$ θ $R'.att_2$*)* is evaluated by the following formula (9):

$$\mu_{v_3} = \max_{t' \in R'} min(\mu_{R'}(t'), \mu_{c_2}(t'), \mu_\theta(t.att_1, t'.att_2)) \tag{9}$$

This interpretation preserves the equivalence between the *exists* operator and the $in_=$ and $in_\approx$ operators, when $\theta \in \{\approx, =\}$ and with the join operator.

In the context of bipolarity, these equivalences hold, and the condition v_3 is rewritten as v_3' = *exists (Select * From* R' *Where* (c_2, s_2) *and* $R.att_1 \theta R'.att_2$*).*

The condition v_3' is evaluated by the following formula (10):

$$(\mu_{c\_v_3'}, \mu_{w\_v_3'}) = \underset{t' \in R'}{lmax}\, lmin((\mu_{R'_c}(t'), \mu_{R'_w}(t')), (\mu_{c_2}(t'), \mu_{w_2}(t')), (\mu_\theta(t.att_1, t'.att_2), \mu_\theta(t.att_1, t'.att_2))) \tag{10}$$

From the formula (10), we can define the *exist* operator as the retrieval of the greatest couple of grades which satisfies the imbricated query.

4.6 The Extension of the *θany* Operator to Bipolarity

In the SQLf language, a query involving *θany* can be rewritten as a query involving the *exists* operator. This equivalence is also valid in the Bipolar SQLf

language. As consequence, the two following queries are equivalent:
Q_7: *Select A From R Where* (c_1, w_1) *and* $att_1\,\theta\,any$
(Select att_2 *From* R' *Where* (c_2, w_2)*);*
Q_8: *Select A From R Where* (c_1, w_1) *and exists*
(Select att_2 *From* R' *Where* (c_2, w_2) *and* $R.att_1 \theta R'.att_2$*);*

The evaluation of a θany query relies then on the formula (10).

4.7 Extension of Aggregate Functions Based Query to Bipolarity

Aggregate functions such as *sum*, *count*, *avg*, *min*, *max* are used to perform arithmetic operations over a set of tuples. The following query expresses the main form of an SQLf query based on aggregate functions:

Q_9: *Select A From R Where c Group By A Having*
$c_{f_1}(agg_1(att_1))$ *cnt... cnt* $c_{f_n}(agg_n(att_n))$*;*

where c is a boolean condition, $agg_1, ..., agg_n$ are aggregate functions which are applied resp. on attributes $att_1, ..., att_n$. The returned values are, then, used as parameters for fuzzy conditions $c_{f_1}, ..., c_{f_n}$ to determine their grades of satisfaction. Finally, the obtained grade are combined depending on connectors *cnt*.

The same principal of partitioning is used in the case of fuzzy bipolar conditions. The following query is the main form of such a partitioning:

Q'_9: *Select A From R Where c Group By A Having*
$(c_1(agg_1(att_1)), w_1(agg_1(att_1)))$ *cnt ... cnt* $(c_n(agg_n(att_n)), w_n(agg_n(att_n)))$*;*

where *cnt* can be either an *and* or an *or* operator. The combination of the couples of grades returned by $(c_i, w_i), i = 1...n$ is based on $lmin$ and/or $lmax$ operators.

5 Conclusion

This article has considered the definition of an SQL-like language to express preferences defined by fuzzy bipolar conditions. Such fuzzy bipolar conditions are extension of fuzzy conditions. This language (namely Bipolar SQLf language) is an extension of the SQLf language to bipolarity. It is based on a relational bipolar algebra that defines basis operators and provides a well appropriate interpretation for each language statement. In this article, we have defined basic statements (projection, selection, join, etc.) and nesting operators such as $in_=$, $in_\approx$, $exists$ and θany.

As future works, we aim at extending the language to fuzzy bipolar quantified propositions, to be able to express queries based on linguistic quantifiers such as *"find stores that have* ***most of*** *their sellers are young* ***and if possible*** *well paid"*, and to study queries corresponding to divisions involving fuzzy bipolar relations, such as *"find students who are well scored* ***and if possible*** *very well scored in all difficult* ***and if possible*** *in all very difficult courses"*. An implementation of a prototype for query evaluation is in progress, and in order to provide users with personalized services, this language is intended to be integrated into a platform of flexible querying of heterogeneous and distributed

information systems developed in the field of multimodal transportation networks, in which complex queries could be expressed such as: *"find journeys from Lannion to Brussels which are fast and having early departures,* ***and if possible*** *not expensive and having steps which go through stations in which are located good restaurants which serve health foods* ***and if possible*** *not expensive".*

Acknowledgments

We warmly thank the Brittany region, the department of Côtes-d'Armor and the National Agency for Research (AOC Ref. ANR-08-CORD-009) for financing this work.

References

1. Bosc, P., Liétard, L., Pivert, O., Rocacher, D. Base de données - Gradualité et imprécision dans les bases de données Ensembles flous, requêtes flexibles et interrogation de données mal connues, 1 ed. Technosup. 2004.
2. Bosc, P., Pivert, O. SQLf: A relational database langage for fuzzy querying. IEEE Transactions on Fuzzy Systems 3, 1 (Feb 1995), 1–17.
3. Bosc, P., Pivert, O., Liétard, L., Mokhtari, A. Extending relational algebra to handle bipolarity. In 25th ACM Symposium on Applied Computing, SAC'10 (2010), pp. 1717–1721.
4. Bouchon-Meunier, B., Dubois, D., Godo, L., Prade, H. Fuzzy sets in approximate reasoning and information systems. The Handbook of fuzzy sets. Kluwer Academic Publishers, 1999, ch. 1 : Fuzzy set and possibility theory in approximate and plausible reasoning, pp. 27–31.
5. Chomicki, J. Querying with intrinsic preferences. In In Proceedings of the 8th International Conference on Extending Database Technology (2002), pp. 34–51.
6. Dubois, D., Prade, H. Bipolarity in flexible querying. LNAI 2522, 174–182. 2002
7. Dubois, D., Prade, H. Bipolarité dans un processus d'interrogation flexible. In Rencontres francophones sur la Logique Floue et ses Applications, LFA (2002).
8. Kiebling, W. Foundation of preferences in database systems. In Proceedings of the 28th VLDB Conference, Hong Kong, China (2002).
9. Li, C., Chang, K. C.-C., Ilyas, I. F., Song, S. RankSQL: Query algebra and optimization for relational top-k queries. In SIGMOD, Baltimore, Maryland, USA. (Jun 2005), ACM, Ed.
10. Liétard, L., Rocacher, D. On the definition of extended norms and co-norms to aggregate fuzzy bipolar conditions. In IFSA/EUSFLAT (2009), pp. 513–518.
11. Liétard, L., Rocacher, D., Bosc, P. On the extension of SQL to fuzzy bipolar conditions. In The 28th North American Information Processing Society Annual Conference (NAFIPS'09) (2009).
12. Tré, G. D., Zadrozny, S., Matthé, T., Kacprzyk, J., Bronselaer, A. Dealing with positive and negative query criteria in fuzzy database quering bipolar satisfaction degrees. LNAI, FQAS 5822 (2009), 593–604.
13. Zadeh, L. Fuzzy sets. Information and control 8, 3 (1965), 338–353.
14. Zadrozny, S., Kacprzyk, J. Bipolar queries using various interpretations of logical connectives. LNAI, IFSA 4529 (2007), 182–190.
15. Zadrozny, S., and Kacprzyk, J. Bipolar queries: An approach and its various interpretations. In Proceedings of IFSA/EUSFLAT (2009), pp. 1288–1293.