Extending databases to precision-controlled retrieval of qualitative information

Victor Polo de Gyves,[1] Adolfo Guzman-Arenas[1,2] and Serguei Levachkine[2]

(1) SoftwarePro International; (2) Centro de Investigación en Computación, Instituto Politécnico Nacional. 07738 Mexico City, MEXICO
degyves@gmail.com, a.guzman@acm.org, sergei@cic.ipn.mx

Abstract. A hierarchy is an arrangement of qualitative values in a tree with certain properties. Hierarchies allow to define the confusion *conf(r, s)* in using qualitative value r instead of the intended or correct value *s*. From here, "predicate P holds for object *o*", written P(*o*), is generalized to "P holds for *o* within confusion ε", written $P_\varepsilon(o)$. These *precision-controlled predicates* are useful to retrieve approximate answers, where the error (confusion) is known.

The predicates are implemented through an extended SQL that uses *confusion* to retrieve information from a database. We show how to extend *any* database for precision-controlled retrieval. Limiting the total error is also useful, and this is achieved by predicate P^{ε}. Examples are given.

1. Introduction and related work

A datum makes sense only within a context. Intuitively, we know that "computer" is closer to "office" than to "ocean" or to "dog." A "cat" is closer to "dog" than to "bus station." "Burning" is closer to "hot" than to "icy." How can we measure these similarities?

A hierarchy describes the structure of qualitative values in a set S. A **(simple, normal) hierarchy** is a tree with root S and if a node has children, these form a partition of the father [1]. A simple hierarchy describes a hierarchy where S is a set (thus its elements are not repeated not ordered). For example: live being{animal{mammal, fish, reptile, other animal}, plant{tree, other plant}}. In a **percentage hierarchy** [3], the size of each set is known,[1] for instance: AmericanContinent(640M){North America(430M) {USA(300M), Canada(30M), Mexico(100M)} Central America (10M), South America(200M)}. The nodes of a percentage hierarchy are bags (sets where repetition is allowed). In an **ordered hierarchy** [2], the nodes of some partitions obey an ordering relation (they are ordered sets): object{tiny, small, medium, large}*.[2] Finally, a **mixed hierarchy** combines the three former types. Other works on retrieval of approximate answers are referenced in [4].

[1] Notation: after each set we write its size in parenthesis. Here we write number of inhabitants.
[2] Notation: an * is placed at the end of the partition, to signify that it is an *ordered* partition.

For these four types of hierarchies we define *conf(r, s)* as the confusion or error in using value r instead of s, the intended or correct value. These definitions agree with the human sense of estimation in closeness for several wrong but approximate answers to a given question; each is applicable to particular endeavors.

The main trust of the paper is in implementation. We define an extended SQL syntax (XSQL) that deals with approximate queries on elements in a database holding qualitative values hierarchically structured. XSQL expresses precision-controlled predicates (§3). The user writes his queries in XSQL. A program (§4.1) converts an XSQL expression back to (pure) SQL. Another program (§4.2) converts hierarchies into tables (storing confusion values) that are added to the (normal) database. Thus, the extension (to precision-controlled retrieval) of *any* database is possible. Some examples are given, mainly for simple and percentage hierarchies, due to page limit.

2. Confusion in hierarchies

Who wrote *Leaves of Grass?* Walt Whitman is the right answer; Edgar Allan Poe a close miss, Michael Jordan a fair error, and Mexico City or cellphone a gross error. What is closer to a *violin*, a *harp*, a *flute* or a *camel*? Can we measure these errors? Yes, with hierarchies of symbolic values. Some definitions from [1-4] are:

Let H be a simple hierarchy. If r, s ∈ H, then the confusion in using r instead of s, written conf(r, s), is:

- conf (r, r) = conf (r, s) = 0, where s is any ascendant of r; (1)
- conf (r, s) = 1 + conf (r, father_of(s)) ♦ (2)

To measure *conf*, count the descending links from r to s, the intended or correct value. Function *conf* is not a distance, nor an ultradistance. To differentiate from other linguistic terms like relatedness or closeness, we prefer to use 'confusion.' *Example:* in table 8, conf(Florida, USA)=1, conf(USA, Florida)=0, conf(USA, Mexico City)=2.

Let H be an ordered hierarchy. The confusion in using r instead of s, conf'' (r, s), is defined as follows:

- conf'' (r, r) = conf (r, any ascendant of r) = 0;
- If r and s are distinct brothers, conf''(r, s) = 1 if the father is not an ordered set; else, conf''(r, s) = the relative distance from r to s = the number of steps needed to go from r to s in the ordering, divided by the cardinality-1 of the father; (3)
- conf'' (r, s) = 1 + conf''(r, father_of(s)). ♦

This is like conf for *simple hierarchies* (formed by sets), except that in them the error between two brothers is 1, and here it is a number in (0, 1]. *Example*: Let Temp = {icy, cold, normal, warm, hot, burning}*. Then, conf''(icy, cold)= 1/5, while conf'' (icy, burning)=1.

Let H be a percentage hierarchy. Let S be the set at the root of H. The **similarity** in using r instead of s, sim^b (r, s), is:

- sim^b (r, r) = sim^b(r, any ascendant_of (r)) = 1;

- if r is ascendant of s, $sim^b(r, s)$= number of elements of S∩r∩s / number of elements of S∩r = relative popularity of s in r. ♦ [3]

The confusion in using r instead of s, conf'(r, s), is $1 - sim^b(r, s)$. ♦ (4)

Example: If baseball player(9) = {pitcher(1), catcher(1), base player(3){first base (1), second base(1), third base(1)}, field player(3){left fielder(1), center fielder(1), right fielder(1)}, shortstop(1)}, then *(a)* conf'(field player, baseball player)= $1 - sim^b$(fielder, baseball player)= 0; *(b)* conf'(baseball player, field player)= 1 – 1/3 = 2/3; *(c)* conf' (baseball player, left fielder)= 8/9; *(d)* conf' (base player, left fielder) = 2/3. This ends the definitions taken from [1-4].

Let H be a mixed hierarchy. To compute *sim(r, s)* in a mixed hierarchy:

- apply rule (1) to the *ascending* path from *r* to *s*;
- in the descending path, use rule (3) instead of rule (2), if p is an ordered set;[4] or use rule (4) instead of (2), when sizes of p and q are known. ♦ That is, use (4) instead of (2) for percentage hierarchies.

This definition is consistent with and reduces to previous definitions for simple, ordered, and percentage hierarchies.

3. Querying a database with predicates that are imperfectly fulfilled

Precision-controlled predicates. A powerful use of *confusion* is to define predicates over objects having attributes with domains on hierarchies, and to define some "looseness of fit" for these predicates. That is, a predicate P shall be satisfied within a given confusion [1]. Let *Hv* stand for a hierarchical variable, and *v* its value for object *o*. We define predicate P_ε thus [1]:

P holds for object o with *confusion* ε (written P_ε *holds for o*, or $P_\varepsilon(o)$) if:

- When *P* is of the form: *(Hv = s),* iff conf(v, s) ≤ ε. (footnote [5])
- When *P* is of the form *P1* ∨ *P2,* iff *P1* holds for *o* or *P2* holds for *o*.
- When *P* is of the form *P1* ∧ *P2,* iff *P1* holds for *o* and *P2* holds for *o*.
- When *P* is of the form ¬*P1,* iff *P1* does not hold for *o*.♦

Examples (Figs. 2, 4 and 5): the predicate $(address = North\_America)_0$ will match any person living in North America or any of its regions (subsets). The predicate $(address = Mexico\ City)_1$ will match any person living in Mexico City, Jalisco, Guadalajara or Mexico. The predicate $(address = Mexico\ City \vee industrial\_branch = Mexican\ food)_1$ is equal to $(address = Mexico\ City)_1 \vee (industrial\_branch = Mexican\ food)_1$ = {Garcia Productores, Mole Doña Rosa} ∪ {Luigi's Italian Food, Mole Doña Rosa}. The predicate $(address = Mexico\ City \wedge industrial\_branch = Mexican\ food)_1$

[3] Relative popularity or percentage of s in r = number of elements of S that are in r and that also occur in s / number of elements of S that are also in r.

[4] Here, p and q are two consecutive elements in the path from r to s, where q immediately follows p. That is, r → …p→q… →s.

[5] That is, the value *v* of property *Hv* for the object *o* can be used instead of *s* with confusion *ε*.

is equal to *(address = Mexico City)$_1$ ∧ (industrial_branch = Mexican food)$_1$* = {Mole Doña Rosa}.

From the definition of P_ε *holds for o,* it is true that $(P \vee Q)_\varepsilon = (P_\varepsilon \vee Q_\varepsilon)$. This means that for $(P \vee Q)_a = (P_b \vee Q_c)$, a = min(b, c). Similarly, for $(P \wedge Q)_a = (P_b \wedge Q_c)$, we have a = max(b, c).

In addition, we define a predicate with "delimited" confusion ε if the sum of the partial confusions is ≤ ε, thus:

P holds for object *o*, but **delimited by ε** [read P^ε *delimited by* ε*, holds for o;* written $P^\varepsilon(o)$], when *P* is of the form *P1∧ P2∧...Pk* and ∃ε1,ε2,… εk≥0 such that $P1_{\varepsilon1}$(o) and $P2_{\varepsilon2}$(o)… and $Pk_{\varepsilon k}$(o) and ε1+ε2+…+εk≤ε. ♦ P "delimited by ε" means that the accumulated confusions should not exceed ε. Note that the "delimited" confusion does not apply to disjunctive predicates (of type P1∨P2∨…), because these hold even when only one Pi holds, and therefore it does not make sense to add the confusion of the Pi's not holding. Example (Figs. 2, 4 and 5): *(address = Mexico City ∧ industrial branch = computer)*^{1} = {Garcia Productores} because, for each of the customers of Fig. 2, the accumulated confusion is, respectively, 2+0, 0+0.7, 2+0.7, 2+0, 2+0.7, 2+0.7, 1+0.7, 2+0.7, 2+0.7, 2+0.7.

3.1 Extended SQL (XSQL)

To query with controlled precision a table T of a database, SQL is extended by these constructs:

- `conf(R,s)`≤ε, an XSQL representation for $(R=s)_\varepsilon$, is a condition procedure used in a `WHERE` or `HAVING` clause, which is true iff conf(r, s)≤*ε.* R is the name of a column of T that is a hierarchical variable (a variable or column having hierarchical values), *r* is each of these values, and *s* is the intended or expected qualitative value. ♦ *Example*: `conf(address,mexico)`≤0 represents in XSQL the predicate *(address = mexico)*$_0$ and will select all rows from figure 2 whose address is Mexico with confusion 0; that is, all rows where *(address = r)* and conf(r, mexico)≤ 0. It will return rows 2 and 7.
- `conf(R)` is an XSQL expression [a shorthand for `conf(R,s)`], used in '`SELECT conf(R)`', or '`GROUP BY conf(R)`' or '`ORDER BY conf(R)`', which returns for each row of table T, the value conf(R,s). ♦ That is, `conf(R)` returns for table T a list of numbers corresponding to the confusion of the value of property R for each row of T. *Example*: see figure 3.

3.2 The user writes a query EXPR in XSQL when he has P_ε or P^ε in mind

The algorithm `EXPR`= replace(P) to substitute (the user thinks about precision-controlled predicate P and writes `EXPR` instead) predicate P by its equivalent XSQL expression `EXPR` is:

- $(R = s)_\varepsilon$ should be replaced by '`conf(`' R '`,`' s '`)≤`' ε, when R is the name of a column of a table, and s a symbolic value.
- $(P1 \vee P2)_\varepsilon$ should be replaced by '`(`' replace($P1_\varepsilon$) '` OR `' replace ($P2_\varepsilon$) '`)`'.
- $(P1 \wedge P2)_\varepsilon$ should be replaced by '`(`'replace($P1_\varepsilon$) '` AND `' replace ($P2_\varepsilon$)'`)`'.
- $\neg P$ should be replaced by '`NOT (`' replace (P) '`)`'.
- $(P1 \vee P2)^\varepsilon$ should be replaced by '`(`'replace($P1$) '` AND `' replace($P2$) '` AND (conf(`' $P1$ '`)+conf(`' $P2$ '`))≤`' ε'`)`'. ♦

Example: (industrial branch = food)$_0$ ∧ [(address = pasadena) ∨ (address = mexico city)]$_1$ is replaced by `conf (industrial_branch, food)≤ 0 AND (conf(address, pasadena)≤ 1 OR conf (address, mexico city)≤ 1)`. *Example*: *(address = Mexico City ∧ industrial branch = computer)*^{1} is replaced by `(conf(address, Mexico City)≤1 AND conf (industrial_branch, computer) ≤1 AND (conf(address)+conf (industrial_branch))≤1)`.

3.3 Queries: retrieving objects that match P_ε

Example: *(address = usa)*$_1$ becomes the XSQL query `conf(address, usa)≤1`, which returns any object whose value of property `address` can be used instead of `usa` with confusion 1. *Example*: Figure 1 shows customers (of figure 2) for which *(address = california)*$_1$. This returns every record, except for Mole Doña Rosa [its address is somewhere in Mexico and conf(mexico, california)=2, by figure 4]; except for Garcia Productores [its address is in Mexico City and conf (mexico city, california)=2]; except for Luigi's Italian food [its address is somewhere in North America and conf(north america, california)=2]; except for Canada seeds [because conf(canada, california)=2]. Figure 3 sorts the result set based on the confusion.

Fig. 1. Querying conf(address, california)$_1$: any customer in California with confusion 1

```
select customer.name, customer.address
from customer
where conf(customer.address,'california')<=1

NAME             ADDRESS
East coast meat  florida
Media Tools      new york
Tom's Hamburgers pasadena
Microsol         silicon valley
Tampa tobacco    tampa
Texas fruits     texas
```

Fig. 2. Table of customers

name	industrial_branch	address	discount
Media Tools	computers	new york	0
Garcia Productores	tequila	mexico city	0
Tom's Hamburgers	food	pasadena	0
Microsol	software	silicon valley	0
East coast meat	meat	florida	0
Luigi's italian food	italian food	north america	0
Mole Doña Rosa	mexican food	mexico	0
Texas fruits	fruits	texas	0
Tampa tobacco	cigars	tampa	0
Canada seeds	food	canada	0

Fig. 3. Querying, sorting and showing values for conf(address, california)$_1$

```
select customer.name, customer.address,
conf(customer.address) from customer where
conf(customer.address,'california')<=1
order by conf(customer.address)

NAME             ADDRESS        CALIFORNIA
Tom's Hamburgers pasadena       0
Microsol         silicon valley 0
Media Tools      new york       1
Tampa tobacco    tampa          1
Texas fruits     texas          1
East coast meat  florida        1
```

Fig. 4. The addresses of customers form a simple hierarchy. Hierarchies are used in §4.2 to generate confusion tables such as that of figure 7

```
Property: address;
hierarchy: world{
                  north_america{
                     usa{
                       california{
                           silicon valley,
                           pasadena, },
                       new york{
                           new york city
                       },
                       florida{
                           miami,
                           tampa
                       },
                       texas
                     }
                     canada,
                     mexico{
                       mexico city,
                       jalisco{
                           guadalajara } } } }
```

Fig. 5. Mixed hierarchy of industrial branch for customers, using percentage values. The percentage values represent the products consumed in a business organization

```
Property: industrial branch;
hierarchy:
industrial branch(1){
    computer(.3){
        software(.12),
        hardware(.18)
    },
    human consumption(.7){
        food(.56){
            prepared food(.112){
                mexican food(.0448),
                italian food(.0672)
            },
            meat(.168),
            fruits(.28)
        }
        drinks and cigars(.14){
            drinks(.056){
                whiskey(.0112),
                beer(.028),
                tequila(.0168)
            },
            cigars(.084) } } }
```

Precision-controlled retrieval in percentage hierarchies can also be done. *Example:* Give me the customers whose industrial branch (figure 5) is food (with confusion of 1), sort and show the confusion values. Results are in figure 6.

Fig. 6. Querying, sorting and showing values for *(industrial_branch = food)*$_1$ for customers of figure 2.

```
select customer.name, customer.industrial_branch,
conf(customer.industrial_branch) from customer
where conf(customer.industrial_branch,'food')<=1 order by
conf(customer.industrial_branch)

NAME                 INDUSTRIAL_BRANCH FOOD
Luigi's italian food italian food      0
Tom's Hamburgers     food              0
Canada seeds         food              0
Texas fruits         fruits            0
East coast meat      meat              0
Mole Doña Rosa       mexican food      0
Garcia Productores   tequila           0.44
Tampa tobacco        cigars            0.44
Microsol             software          0.44
```

4. Implementation of confusion-controlled queries

An *extension kit* permits *any database* to handle imprecise retrieval: First, a parser is used to translate (§4.1) XSQL predicates to a (pure) SQL query able to use pre-calculated *confusion* tables (called SCTs). Then, execution of the new SQL predicate is carried out. §4.2 explains how these tables are created.

4.1 Translating (by the parser) XSQL queries to valid SQL queries

Since we extended SQL by adding only conf(R, s) and conf(R), we need to deal only with these two. If S is a valid XSQL `SELECT` query containing `conf(R,s)`≤ε or `conf(R)`:

1. Let t(R) = R*'s* table name and r(R) = R*'s* column name.
2. Add t(R) to the list of tables (`FROM` clause).
3. Translate any `conf(R,s)`≤ε as '`(confusion.`' t(R) '`_`' r(R) '`norm.name=`' t(R) '`.`' r(R) '` AND confusion.`' t(R) '`_`' r(R) '`_norm.`' s '`≤`' ε '`)`'.
4. Translate any `conf(R)` as '`confusion.`' t(R) '`_`' r(R) '`_norm.`' s.

Example: `select customer.name from customer where conf(customer.industrial_branch, 'food') <=1` translates to

```
select customer.name from customer,
confusion.customer_industrial_branch_norm where
(confusion.customer_industrial_branch_norm.name =
customer.industrial_branch AND
confusion.customer_industrial_branch_norm.food <=1).
```

If S is a valid XSQL `UPDATE/DELETE` arising from a predicate using P^{ε} or P_{ε} and containing `conf(R,s)`≤ε only in the `WHERE` section:

1. Let t(R) = R*'s* table name and r(R) = R*'s* column name.
2. Create a valid SQL `SELECT`, named S2= '`(SELECT `' R '` FROM `' t(R) '` WHERE conf(`' R '`.`' r(R) '`)≤`' ε '`)`'.
3. Translate S2 [to get rid of the conf(R.r(R))<ε], generating a new expression S2'.
4. Replace every appearance of `conf(R,s)`≤ε in S by S2'.

For `INSERT` sentences, no confusion is valid, except for `INSERT…SELECT`. In that case, translate the `SELECT` part as described.

4.2 Implementing the calculation of confusion values in databases

Applying *conf (R, s)*≤ε in a database table *T* involves the use of a function *f* that, for every record $x \in T$, takes the *r* value stored in the *R* property of *x* and calculates *conf (r, s)*. If less or equal to ε, the record is returned. Doing this calculation for every query is slow. Instead, sets of pre-calculated database tables of confusion (SCT's) are used to speed up the query process. This process is shown (displayed) in figure 7.

Fig. 7. Steps to compute a confusion table (CT) for a table T. The example uses for T the table of Fig. 2, and the hierarchies in figs. 4 and 5. Two CT's are produced, one is shown in Fig. 8

Step	*Example*
Let *T* be a database table. $Tp=\{a_1 \ldots a_n\}$ are properties of *T*; $Th=\{a_m \ldots a_n\}$ are hierarchical properties of *T* and $Th \subseteq Tp$.	T = customer (see table customer in Fig. 2) Tp = {name, industrial_branch, address, discount} Th = {industrial_branch, address}
Consider each $a_x \in Th$, *and form* $H=\{ h_x \mid h_x$ *is the hierarchy corresponding to* $a_x\}$	H = $\{h_{industrial\_branch}, h_{address}\}$ = the set of hierarchies of T.
Let $CT = \{confusion.T\_a_m\_norm, \ldots, confusion.T\_a_n\_norm\}$ where $a_x \in Th$. For each element $e_x \in CT$, a precalculated confusion table will be created, as described later.	CT = {confusion**.**customer_industrial_branch_norm, confusion**.**customer_address_norm}
Given an element $e_x \in CT$, the set of properties *sp* for e_x is a function defined as: $sp(e_x)= \{$'name', $h_{x1} \ldots h_{xn}\}$ where h_{xy} is an element in the hierarchy h_x and $h_x \in H$.	sp(confusion**.**customer_industrial_branch_norm)={name, industrial_branch, computer, human consumption, software, hardware, food, drinks and cigars, prepared food, meat, fruits, drinks, cigars, mexican food, italian food, whiskey, beer, tequila } sp(confusion**.**customer_address_norm)={name, world, north america, canada, usa, mexico, california, new york, florida, texas, mexico city, jalisco, silicon valley, pasadena, new york city, miami, tampa, guadalajara}

Given an element $e_x \in CT$, the set of objects *so* for e_x is a function defined as: $so(e_x)= \{ \{h_{x1}, conf(h_{x1}, h_{x1}) \ldots conf(h_{x1},h_{xn}) \}, \ldots \{h_{xn}, conf(h_{xn},h_{x1}) \ldots conf(h_{xn}, h_{xn}) \} \}$, where h_{xy} is an element in the hierarchy h_x and $h_x \in H$.	so(confusion.customer_industrial_branch_norm)={
	{industrial_branch,0.0,0.6,0.4,0.8,0.7,0.5,0.8,0.9,0.8,0.7,0.9,0.9,0.9, 0.9,0.9,0.9,0.9}, {computer,0.0,0.0,0.4,0.8,0.7,0.5,0.8,0.9,0.8,0.7, 0.9, 0.9,0.9,0.9,0.9,0.9,0.9,0.9},
	...
	{tequila,0.0,0.6,0.0,0.8,0.7,0.5,0.0,0.9,0.8,0.7,0.0,0.8,0.9,0.9,0.9,0.8,0.0}
	} *(the complete set of objects is not shown)*
	so(confusion.customer_addres_norm)={
	{world, 0 ,1 ,2 ,2 ,2 ,3 ,3 ,3 ,3 ,3 ,3 ,4 ,4 ,4 ,4 ,4 ,4},
	{north america,0 ,0 ,1 ,1 ,1 ,2 ,2 ,2 ,2 ,2 ,2 ,3 ,3 ,3 ,3 ,3 ,3},
	...
	{guadalajara, 0 ,0 ,1 ,1 ,0 ,2 ,2 ,2 ,2 ,1 ,0 ,3 ,3 ,3 ,3 ,3 ,0}
	} *(the complete set of objects is not shown)*

Finally, given an element $e_x \in CT$, the SQL confusion table for e_x is an SQL table defined as $sp(e_x)$ and filled with elements in $so(e_x)$; the name of the table will be e_x. $SCT=\{e_m, \ldots, e_n\}$ is the set of confusion tables for *T*.	

Fig. 8. SQL confusion table for `confusion.customer_address_norm`, as generated by the algorithm of figure 7. Each row is r while each column is s. Another confusion table (for `confusion.customer_industrial_branch_norm`, not shown) is generated, too. Thus, the set SCT generated in figure 7 contains these two confusion tables (CT).

name	world	north america	canada	usa	mexico	california	new york	florida	texas	mexico city	jalisco	silicon valley	pasadena	new york city	miami	tampa	guadalajara
world	0	1	2	2	2	3	3	3	3	3	3	4	4	4	4	4	4
north ame	0	0	1	1	1	2	2	2	2	2	2	3	3	3	3	3	3
canada	0	0	0	1	1	2	2	2	2	2	2	3	3	3	3	3	3
usa	0	0	1	0	1	1	1	1	1	2	2	2	2	2	2	2	3
mexico	0	0	1	1	0	2	2	2	2	1	1	3	3	3	3	3	2
california	0	0	1	0	1	0	1	1	1	2	2	1	1	2	2	2	3
new york	0	0	1	0	1	1	0	1	1	2	2	2	2	1	2	2	3
florida	0	0	1	0	1	1	1	0	1	2	2	2	2	2	1	1	3
texas	0	0	1	0	1	1	1	1	0	2	2	2	2	2	2	2	3
mexico cit	0	0	1	1	0	2	2	2	2	0	1	3	3	3	3	3	2
jalisco	0	0	1	1	0	2	2	2	2	1	0	3	3	3	3	3	1
silicon vall	0	0	1	0	1	0	1	1	1	2	2	0	1	2	2	2	3
pasadena	0	0	1	0	1	0	1	1	1	2	2	1	0	2	2	2	3
new york c	0	0	1	0	1	1	0	1	1	2	2	2	2	0	2	2	3
miami	0	0	1	0	1	1	1	0	1	2	2	2	2	2	0	1	3
tampa	0	0	1	0	1	1	1	0	1	2	2	2	2	2	1	0	3
guadalajar	0	0	1	1	0	2	2	2	2	1	0	3	3	3	3	3	0

4.3 Two examples using confusion with mixed hierarchies for retrieval of information in databases

In example 4.3.1 we need to hire one computer specialist (from candidates of figure 10) with some special characteristics. The problem is to select the best candidate fulfilling the desired requirements. It would be good to use confusion, specially if there are few or no candidates covering all of the requirements. Figures 9, 11, 12 and 13 define the hierarchies. In example 4.3.2 we update groups of objects that depend on hierarchical structures.

Fig. 9. Hierarchy for `degree`. It is a simple hierarchy, as defined in §1

```
degree{
    mathematics{
        computer{high school computer, college computer},
        physics college,
        electric{electric college, electronics college, electric high
                 school},
    }
    biology{medical, biologist},
    humanities{history college, languages college}
}
```

Fig. 10. Candidates and their qualifications. The ideal candidate should have a college degree related to computers; should know Java; should have worked previously as system analyst, and should have experience in Solaris

NAME	DEGREE	LANG	PREVIOUSWORK	OPERATINGSYSTEM
Alfred	college_computer	java	system_analyst	unix
Brenda	high_school_computer	basic	hackman	windows_xp
John	high_school_computer	prolog	secretary	none
Thomas	electronics_college	python	programmer	linux
Susan	electronics_college	basic	programmer	windows_2000
Abraham	electric_college	cpp	operator	bsd
Natalie	electric_high_school	pascal	secretary	none
Martin	physics_college	lisp	manager	solaris
Alex	physics_college	lisp	programmer_leader	windows_2000
Ernest	college_computer	shell	operator	linux
Ann	history_college	none	other	none
Sam	biologist	none	other	linux
Fred	languages_college	prolog	manager	macos9
Robert	electric_college	java	database_administrator	linux
Bill	high_school_computer	cpp	other	solaris

Fig. 11. Hierarchy for `work`, pertinent to column `PREVIOUSWORK` of figure 10

```
work{
    computers related{programmer, programmer leader, system
        analyst, database administrator},
    administration related{chief executive, manager},
    operations related{operator, secretary, hackman}
    other
}
```

4.3.1 Limiting the errors.

Sort candidates to hire. Here we analyze the use of P^{ε}. It is a way to limit the total error in predicates: A list of candidates is given to cover available jobs on a new enterprise. The requirements for the job is given in a predicate P. While P_{ε} can select and sort this list for a given value of ε, it is better to limit the sum of confusions produced by each candidate (due to imperfect match of his qualifications). This is accomplished by P^{ε}, where ε is such limit. We will use figures 9-13. We begin by using P = (degree = 'high school computer')$_2$ ∧ (programming language = 'java')$_2$ ∧ (previous work = 'system analyst')$_2$ ∧ (experience = 'solaris')$_3$.

Fig. 12. Hierarchy for `programming language`

```
languages{
    programming languages{
        artificial intelligence{prolog, lisp},
        object oriented{cpp, java, python},
        other{pascal, basic, shell}
    },
    none
}
```

We may reduce the *confusion* in P by querying with small ε's, such as in P2=

(degree = 'high school computer')$_1$ ∧ (programming_language = 'java')$_2$ ∧ (previous_work = 'system analyst')$_1$ ∧ (experience = 'solaris')$_2$, so that now only *Alfred* fulfills P. But, what if we need more employees to hire? This is where P^{ε} goes. It allows to use the sum of confusion values to delimit the objects that P holds for. So, we can delimit P of figure 15 as follows: P^5= ((degree = 'high school computer')$_2$ ∧ (programming_language = 'java')$_2$ ∧ (previous_work = 'system analyst')$_2$ ∧ (experience = 'solaris')$_3$)^5. Having done this, now we are controlling the list of resulting objects using the *sum of confusions* (figure 14):

Fig. 13. Hierarchy for `operating systems`.

```
systems{
    operating systems{
        unix{  linux, solaris, bsd}
        microsoft{windows 2000, windows nt 4, windows xp}
        apple{macos9, macosx}
    }
    none
}
```

Fig. 14. Querying with P^{ε} to use the *sum of confusions* as a way to have a control of the returning objects. The column showing the sum of confusion values for each object was added by hand: it is not part of the query.

```
select candidates.name from candidates where
conf(candidates.degree,'high_school_computer')<=2 AND
conf(candidates.programminglanguage,'java')<=2 AND
conf(candidates.previouswork,'system_analyst')<=2 AND
conf(candidates.operatingsystem,'solaris')<=3 AND
conf(candidates.degree)+conf(candidates.programminglanguage)+conf
(candidates.operatingsystem)+conf(candidates.previouswork) <= 5"
NAME    CONF_SUM
Alfred  2
Robert  4
Thomas  5
Bill    3
```

4.3.2 Update.

Give a discount to customers having `food` as `industrial branch`. *(Update)*. Hierarchies in figures 4 and 5 are used, where customers (figure 2) buy from a supermarket. The supermarket wishes to give a discount of 7% to customers related to food, because another supermarket is trying to have these customers. It is possible to do this update using pure SQL, but it involves the execution of several SQL sentences. Using XSQL to update objects delimited by hierarchical qualitative values provides a simpler and faster way to execute to the database server. Use

```
update customer set discount=0.07 where customer.name in
conf(customer.industrial_branch,'food')<=0
```

The 'confusion way' is efficient because the update is solved by filtering the customer's table using joins with SCT's tables. The update result appears in figure 15.

Fig. 15. Update results to customers related to food in the `INDUSTRIAL_BRANCH` property; the sentence was `update customer set discount=0.07 where customer.name in conf(customer.industrial_branch,'food')<=0`

NAME	INDUSTRIAL_BRANCH	DISCOUNT
Media Tools	computers	0.0
Garcia Productores	tequila	0.0
East coast meat	meat	0.07
Luigi's Italian food	italian food	0.07
Mole Doña Rosa	mexican food	0.07
Texas fruits	fruits	0.07
Tampa tobacco	cigars	0.0
Canada seeds	food	0.07
Tom's Hamburgers	food	0.07
Microsol	software	0.0

5. Conclusions

The similarity among symbolic values that form hierarchies is exploited through use of confusion. Predicates with controlled precision $P_\varepsilon(o)$ (called "P holds for o with precision ε") and $P^\varepsilon(o)$ (called "P delimited by ε, holds for o") allow us to define precision-controlled retrieval of hierarchical values. These predicates permit "loose retrieval" (retrieval with defined confusion bounds) of objects that sit in a relational database. Moreover, such database could be an existing "normal" database (where no precision-controlled retrieval was possible), to which one or more definitions of hierarchies are attached. This in fact provides a procedure (a "kit") to extend *any* (existing) database to another in which imprecise retrievals are possible. Furthermore, this extension can be done without recompiling application programs. Old programs (with no precision retrieval) still work as before, whereas new application programs can exploit the database with precision-controlled queries. Thus, a "normal" database now becomes a "precision-controlled" database when the kit is applied to it.

Acknowledgments. Work was partially supported by CONACYT Grant 43377. First and second authors have a SNI National Scientist Award.

References

1. A. Guzman-Arenas and S. Levachkine. Hierarchies Measuring Qualitative Variables. *Lecture Notes in Computer Science* LNCS **2945** (Computational Linguistics and Intelligent Text Processing), (Springer-Verlag 2004). 262-274.
2. A. Guzman-Arenas and S. Levachkine. Graduated errors in approximate queries using hierarchies and ordered sets. *Lecture Notes in Artificial Intelligence* LNAI **2972**, (Springer-Verlag 2004). 139-148. ISSN 0302-9743
3. S. Levachkine and A. Guzman-Arenas. Confusion between hierarchies partitioned by a percentage rule. Unpublished manuscript.
4. S. Levachkine and A. Guzman-Arenas. Hierarchy as a new data type for qualitative variables. Submitted to *Data and Knowledge Engineering*.