J Intell Inf Syst (2006) 26: 59–73
DOI 10.1007/s10844-006-5451-1

# Querying a summary of database

**W. A. Voglozin · G. Raschia · L. Ughetto · N. Mouaddib**



**Abstract** For some years, data summarization techniques have been developed to handle the growth of databases. However these techniques are usually not provided with tools for end-users to efficiently use the produced summaries. This paper presents a first attempt to develop a querying tool for the SAINTETIQ summarization model. The proposed search algorithm takes advantage of the hierarchical structure of the SAINTETIQ summaries to efficiently answer questions such as "how are, on some attributes, the tuples which have specific characteristics?" Moreover, this algorithm can be seen both as a boolean querying mechanism over a hierarchy of summaries, and as a flexible querying mechanism over the underlying relational tuples.

**Keywords** Data summarization · Linguistic summaries · Summary querying · Relational database · Flexible querying · Fuzzy labels

## 1. Introduction

In order to handle the growth in size of databases, many approaches have been developed to extract knowledge from huge databases. One of these approaches consists in summarizing data (see for example Cubero et al., 1999; Dubois and Prade, 2000; Kacprzyk, 1999; Lee and Kim, 1997; Raschia and Mouaddib, 2002). However, summarization techniques are usually not provided with tools for end-users to efficiently use the summaries. As a consequence, users have to directly interpret the summaries, which is conceivable with a few summaries only. In other cases, tools are necessary.

In this paper, the structured data summarization model SAINTETIQ developed in our research team (Raschia and Mouaddib, 2002) is considered. SAINTETIQ provides a compact representation of a database, rewriting the tuples by means of linguistic variables (Zadeh, 1975) defined on each attribute, and classifying them in a hierarchy of summaries. The set of summaries, produced by the process, describes the data in a comprehensible form. Thus,

W. A. Voglozin (✉)· G. Raschia · L. Ughetto · N. Mouaddib
Laboratoire d'Informatique de Nantes Atlantique, Université de Nantes, 2 rue de la Houssinière,
BP 92208, 44322 Nantes Cedex 3, France
e-mail: amenel.voglozin@univ-nantes.fr; guillaume.raschia@univ-nantes.fr;
laurent.ughetto@univ-nantes.fr; noureddine.mouaddib@univ-nantes.fr

each summary, expressed with fuzzy linguistic labels, symbolizes a concept that exists within the data.

This paper proposes a querying mechanism for users to efficiently exploit the hierarchical summaries produced by SAINTETIQ. The first idea is to query the summaries using the vocabulary of the linguistic variables defined for the summarization process. Although linguistic terms are used in the expression of queries, the querying process is clearly boolean. Since the querying vocabulary is the one used within the summaries, the linguistic terms have become the attribute values in the summaries and query answers contain these linguistic terms only. Then, basic queries such as "how are, on attribute(s) $A_k$, the tuples which are $d_{i,j}$ on $A_i$" (e.g., "what is the hardness of metals whose fusion temperature is high"), can be answered very efficiently as the querying process relies on boolean operations. Moreover, the algorithm takes advantage of the hierarchical structure of the summaries, which also works as a kind of multidimensional index, in order to obtain the answer more rapidly. The gain is particularly important in case of a null answer, as only a small part of the summaries hierarchy has to be explored, instead of the entire relation.

Querying the summaries as explained above is interesting as it makes it possible to rapidly get a rough idea of the properties of tuples in a relation. In case of null answers, it clearly saves time: examining the top summary of a hierarchy is enough to know there is no answer for an empty result set query. Suppose for instance that one can access several databases. Querying their summaries can allow to rapidly determine which ones are likely to give an answer.

In other cases, a rough answer is often not enough. Thus, the second idea is to query the database through the summaries, which would be done by 1—selecting some summaries from the summary hierarchy w.r.t. criteria specified by the query and 2—retrieving the database records' identifiers from the selected summaries. This process is then related to the "flexible querying of relational databases" trend of research. Indeed, in this case, linguistic terms are used in the expression of queries, and the answer would be composed of tuples from the original relation, ranked according to a degree of satisfaction to the query.

The next section describes flexible queries of databases and their features compared to classical queries. It exposes some earlier works done in this field by other researchers. Section 3 presents an overview of the SAINTETIQ model, briefly depicting the representations of summaries and the different steps of the summary building process. It also highlights the distinctive aspects of our approach. Section 4 thoroughly explains how advantage can be taken from the use of the SAINTETIQ summaries hierarchies in a flexible querying process. Expression of queries, selection of summaries and formation of results are then reviewed.

## 2. Flexible querying of regular databases

A flexible querying process operating on relational databases searches the tuples for adequacy to a query using an extension of a standard language, usually SQL. According to Larsen (1999), a flexible querying process of a database can be divided in three steps: extension of criteria, selection of results and ordering.

The first step uses similarity between values to extend the criteria, i.e. to allow graded semantics for any criterion, which can now express "around 20" instead of being limited to the binary semantics of "equal to 20" or "between 18 and 22". The second step, namely the selection of results, determines which data will participate in the answer to the query. These data are afterwards referred to by the term "results": the set of all results constitute the answer to a query. The last step (ordering) follows from the extension of criteria. It discriminates

among the results on the basis of their relative satisfaction to the graded semantics: a value of 20 is better ranked than a value of 18.

The fuzzy set theory is often used in flexible querying (see Dubois and Prade, 1997) because it provides a formal framework to handle the graduality and vagueness inherent to natural language. The following works, which are representative of the research on flexibility in database querying, exemplify the use of fuzzy sets. They are essentially characterized by a tuple-oriented processing, the possibility to define new terms and especially, the use of satisfaction degrees.

### 2.1. SQLf

The querying language SQLf, proposed by Bosc and Pivert (1994), is an extension of SQL aiming at "introducing fuzzy predicates into SQL wherever possible". An augmentation of both the syntax and semantics of SQL is performed so that most elements of a query can be fuzzified. These elements include operators, aggregation functions, modifiers (very, really, more or less), quantifiers (most, a dozen) as well as general description terms such as young or well-paid.

Different interpretations are possible for the same query, for instance fuzzy sets crisp cardinality or Yager's ordered weighted averaging operators (Yager, 1988). It occurs for each record and yields a grade of membership (of the record to the query) which is used to rank the results. An example of query in SQLf is "**select** 10 `dpt` **from** EMPLOYEE **group by** `dpt` **having** *most-of*(`age` = *young*) **are** *well-paid*" where standard SQL keywords are in bold face, and `dpt` and `age` are attributes from a relation named EMPLOYEE. The query selects the 10 departments which have the best satisfaction of the condition "most of the young employees are well-paid".

### 2.2. FQUERY

FQUERY (Kacprzyk and Zadrożny, 2001) is an integration of flexible querying into an existing database management system, namely Microsoft Access. The system allows queries with vague predicates expressed through fuzzy sets. Queries may contain linguistic quantifiers and attach different levels of importance to attributes. In doing so, the authors try to apply the *computing with words* paradigm and, eventually, deal with linguistic values, quantifiers, modifiers and relations.

FQUERY uses fuzzy sets for the imprecision aspect and performs a syntax and semantics extension of SQL. Linguistic values and quantifiers are represented as fuzzy sets. On the semantics side, the query is considered as a fuzzy set resulting from the combination of fuzzy sets from linguistic values and quantifiers. Accordingly, each record selected by a classical SQL query, has a satisfaction degree used in a ranking step since it indicates how well the record corresponds to the query.

### 2.3. SummarySQL

Developed by Rasmussen and Yager (1997), SummarySQL is a fuzzy query language intended to integrate summaries into a fuzzy query. The language can evaluate the truth degree of a summary guessed by the user. It can also use a summary as a predicate in a fuzzy query.

A summary expresses knowledge about the database in a statement under the form "**Q** objects in **DB** are **S**" or "**Q R** objects in **DB** are **S**". **DB** stands for the database, **Q** is a linguistic

quantifier and **R** and **S** are summarizers (linguistic terms). One can obtain statements like "**most** people in **DB** are **tall**" or "**most tall** people in **DB** are **heavy**".

Predicates (i.e. summaries) and linguistic terms are fuzzy sets in the expression that represents the selection condition. The expression is evaluated for each tuple and the associated truth values are later used to obtain a truth value for the summary. SummarySQL is used to determine whether, or to what extent, a statement is true. It can also be used to search for fuzzy rules.

### 2.4. FSQL

FSQL (Galindo, 1998) is an extension of SQL in which all SQL expressions are valid. The language is available through a client-server architecture in which the server software accesses an Oracle database. SQL is extended to allow flexible conditions using linguistic labels, fuzzy comparison operators, fuzzy constants and many other fuzzy constructs. For instance, a fuzzy quantifier can be specified in the language's *metaknowledge base* by giving the four values that define a trapezoidal possibility distribution. Each condition in a query can be given a threshold that sets the minimum satisfaction degree for the condition. FSQL does not limit flexibility to the SELECT clause. INSERT, UPDATE and DELETE are also supported.

FSQL is remarkable in two aspects. First, the wide variety of flexible constructs that are available and second, it is one of the very few, it not the only, fuzzy extensions of SQL implemented over an existing database management system.

## 3. Querying the SAINTETIQ summaries

The targeting of database records in flexible queries may lead to prohibitive response times when a large number of records is involved, or when subqueries are employed. Waiting for an answer for a long time is frustrating, particularly when the query fails (that is, it has no matching record in the dataset).

Database summaries offer a means for significantly reducing the volume of input for processes that require an access to the database. The response time benefits from the downsizing. Furthermore, for this querying process, performance does not depend on specific combinations of attributes, i.e., whether the attributes are indexed or not, since these summaries are general indexes for the underlying data (Raschia, 2001).

When querying the summaries, the response time gain is made clearly at the expense of a loss of precision in the answer. This is of no importance when only a rough answer is required. This can be the case for instance when querying a medical database for anonymous, statistical information. Indeed, precise information can violate medical confidentiality. The loss of precision is also of no importance when a request only aims at determining the absence of information in a database. As already said, this is the case when one wants to know if a database is likely to answer the query.

When more details about the tuples are needed, querying the summaries is a first step only: the entire set of relevant tuples can be easily retrieved from the answer summaries. The querying mechanism remains efficient, and there is no loss of precision in the answer. However, the loss is in the querying language expressiveness. At present, only the linguistic variables used to build the summaries hierarchy can be used in the expression of the queries. Moreover, as the generated summaries can be considered as one table in the database relational model,

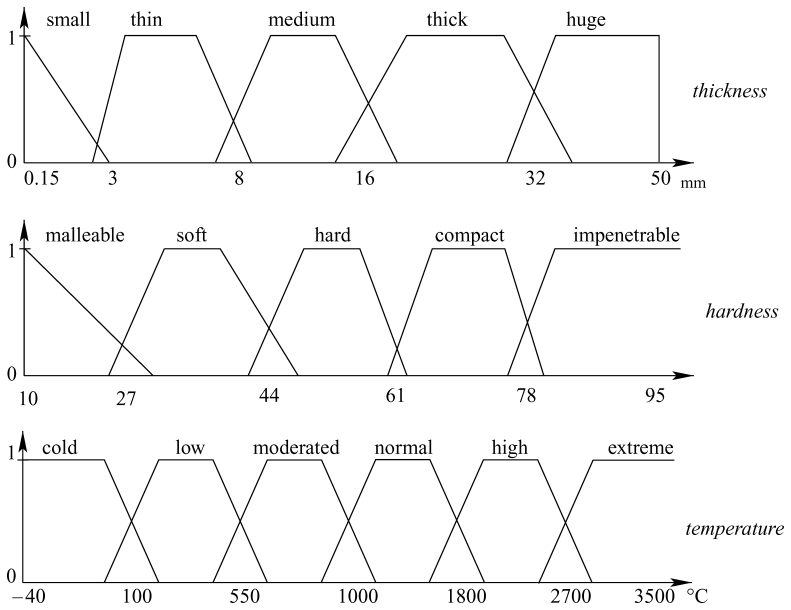


**Fig. 1** Linguistic variables for the MATERIALS table

the implemented querying operations are the selection and projection from the relational algebra.

### 3.1. Running example

In the rest of this paper, a single example illustrates our discussion. It is based on the relation R = (thickness, hardness, temperature) of a table MATERIALS. A tuple from that table describes a material used in an imaginary metallurgy plant to produce square sheets. Attribute thickness is expressed in millimeters and has a limited range (0.15 mm to 50 mm). Attribute hardness is the final product's expected value on scale B of the Rockwell hardness test. Attribute temperature is the melting point for the metal or alloy that constitutes a material.

Figure 1 shows the linguistic variables associated with the attributes of R. These linguistic variables constitute the new attribute domains used for the rewriting of tuples in the summarization process. Table 1 shows the portion of the MATERIALS table that corresponds to the hierarchy in Figure 2. The summaries that appear in Figure 2 are described in Table 2.

### 3.2. Summaries in SAINTETIQ

The SAINTETIQ model aims at apprehending the information from a database in a synthetic manner. This is done through linguistic summaries structured in a hierarchy. The model offers different granularities, i.e., levels of abstraction, over the data. The steps necessary to build a summary hierarchy are described below.

First, records are translated in accordance with a *background knowledge* provided by the user. For each attribute, linguistic variables (which are part of the background knowledge) offer a mapping of the attribute's value to linguistic labels describing that value. For instance,

**Table 1** Part of the MATERIALS table

| Materials | Tuple | Translation |
|---|---|---|
| UZ40 | $t_a = \langle 10, 38, 900 \rangle$ | $t_{a_1} = \langle$ 0.7/medium, 1.0/soft, 0.85/moderated $\rangle$ |
| CuSn12 | $t_b = \langle 8, 40, 850 \rangle$ | $t_{b_1} = \langle$ 0.35/medium, 0.9/soft, 1.0/moderated $\rangle$, $t_{b_2} = \langle$ 0.35/thin, 0.9/soft, 1.0/moderated $\rangle$ |
| CuAs05 | $t_c = \langle 12, 44, 896 \rangle$ | $t_{c_1} = \langle$ 1.0/medium, 0.4/soft, 0.9/moderated $\rangle$, $t_{c_2} = \langle$ 1.0/medium, 0.4/hard, 0.9/moderated $\rangle$ |
| Fe | $t_d = \langle 10, 35, 1530 \rangle$ | $t_{d_1} = \langle$ 0.7/medium, 1.0/soft, 0.85/normal $\rangle$ |
| Ni | $t_e = \langle 5, 35, 1453 \rangle$ | $t_{e_1} = \langle$ 1.0/thin, 1.0/soft, 0.96/normal $\rangle$ |
| ... | ... | ... |

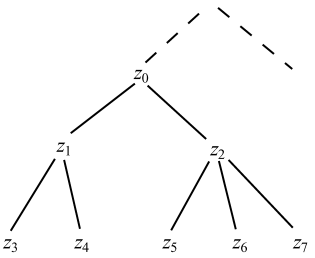


**Fig. 2** Part of a summary hierarchy for MATERIALS

$t_e$.thickness $= 5$ mm is expressed as $t_{e_1}$.thickness $= \{1.0/thin\}$ where 1.0 tells how well the label *thin* describes the value *5 mm* (1.0 is the satisfaction degree of *thin* by *5 mm*). Applying this mapping to each attribute of a relation corresponds to a translation of the initial tuple into another expression called a *candidate tuple*.

One attribute value may be described by more than one fuzzy label (e.g. *8 mm* is described by *medium* and *thin*). It follows that one tuple (for instance $t_b$ and $t_c$ in Table 1) may yield many candidate tuples.

Second, each candidate tuple is incorporated into the growing hierarchy and reaches a leaf node where other candidate tuples with the same labels are stored. This can be seen as a classification of the candidate tuple. It is important to notice that the tree is modified throughout candidate tuples incorporation: it progressively becomes a complete representation of the data. Its evolution is partly controlled by learning operators (see Raschia, 2001) that try to maximize the inner-class similarity and the inter-class dissimiliarity.

An analogy could be that the hierarchy of summaries is a network of pipes with a single entry point at the top and several outlets to a set of buckets at the bottom, one outlet per bucket. Candidate tuples can be seen as objects of different types. From the entry, an object is directed to the bucket it belongs to. But through the process of reaching the adequate bucket, it creates some structural modifications in the network of pipes so some craftsmen (that is, the learning operators) have to constantly adapt the network. Some junctions are made thinner or larger, some pipes are added or deleted, etc... In the end, all objects of the same type are dispatched into buckets and the lengths of paths to the buckets are minimal. Each bucket that

**Table 2** Description of some summaries

| Summary | Intension | Cover |
|---|---|---|
| $z_3$ | ⟨ 1.0/medium, 1.0/soft, 1.0/moderated ⟩ | $t_{a_1}, t_{b_1}, t_{c_1}$ |
| $z_4$ | ⟨ 0.7/medium, 1.0/soft, 0.85/normal ⟩ | $t_{d_1}$ |
| $z_5$ | ⟨ 0.35/thin, 0.9/soft, 1.0/moderated ⟩ | $t_{b_2}$ |
| $z_6$ | ⟨ 1.0/medium, 0.4/hard, 0.9/moderated ⟩ | $t_{c_2}$ |
| $z_7$ | ⟨ 1.0/thin, 1.0/soft, 0.96/normal ⟩ | $t_{e_1}$ |
| $z_1$ | ⟨ 1.0/medium, 1.0/soft, 1.0/moderated + 0.85/normal ⟩ | $z_3, z_4$ |
| $z_2$ | ⟨ 1.0/thin + 1.0/medium, 1.0/soft + 0.4/hard, 1.0/moderated + 0.96/normal ⟩ | $z_5, z_6, z_7$ |
| $z_0$ | ⟨ 1.0/thin + 1.0/medium + 0.7/thick, 1.0/soft + 0.4/hard, 1.0/moderated + 0.96/normal + 0.75/high ⟩ | $z_1, z_2$ |

is not empty at the end after all objects have been through the network of pipes is labeled with a description of its contents.

From the semantic point of view, a summary is a *concept*, a set of records which are similar when rewritten with the terms from the linguistic variables. If the user had to describe the records in the database using the vocabulary they have provided, they would have given the same descriptions many times for different records. Instead of repeating these instances, a summary reflects each description once, keeping track of what records share the same description.

In the hierarchy structure, a level (i.e., a depth in the tree of summaries) can be associated with the relative proportion of data a summary describes: the deeper the summary in the tree (or the lower its level in the hierarchy), the finer the granularity. It follows that the lowest levels (the leaves) contain the most precise and specific summaries. Such summaries can be expressed in the same way as candidate tuples: $z = \langle \alpha_1/d_1, \alpha_2/d_2, \ldots, \alpha_n/d_n, \rangle$. This expression is called the *intension* or *intensional expression* of the summary. It is an important point to note that there is only one label per attribute in the leaf summaries (see $z_3$ in Table 2). Moreover, an initial relational tuple can correspond to several leaves.

By contrast, the root of the tree is the most general summary as it covers all the data. The intensional expression of non-leaf summaries has one or more multi-labeled attributes (e.g. $z_1$ and $z_2$ ). This depends on the labels in candidate tuples covered by the summaries. Thus, in these intermediate-level summaries (the non-leaf nodes of the hierarchy tree), the labels are obtained by just the union of the labels of the children summaries.

## 4. Description of the querying process

As stated in Section 2, the first step of a database flexible querying process consists in extending selection criteria. The linguistic variables used in SAINTETIQ already perform the criteria extension task prior to the actual summarization process. As a consequence, binary inclusion operators can be used to identify the summaries (and also, the relational data)

to be considered as results to a query; there is no need for special operators to deal with the computation-intensive task of extending criteria. This section deals with all aspects of selection from the expression and meaning of a query to its matching with summaries.

### 4.1. Expression of a query

This approach to flexible querying intends to answer questions such as "how are thin materials?" or "how are normal-temperature and soft-hardness materials ?". In the prototype developed for querying, the questions are expressed using a user-friendly interface that composes the corresponding query in an SQL-like language. For the two previous questions, the queries are respectively:

```
Q1: SELECT temperature, hardness FROM MATERIALS
    WHERE thickness IN ("thin")

Q2: SELECT thickness FROM MATERIALS
    WHERE temperature IN ("normal") AND hardness IN ("soft")
```

For a more formal expression of a query, let:

- $S$ be a set of attributes;
- $\mathsf{R}(S)$ be the relation whose tuples are summarized;
- $Q$ be a query (e.g., $Q_1$ and $Q_2$ in the above example);
- $A_i$, $i \in \{1, \ldots, n\}$ be an attribute appearing in the query ($A_i \in S$);
- $d_{i,j}$, $j \in \{1, \ldots, m\}$ be a label (or descriptor) of attribute $A_i$, also appearing in the query.

In a query, descriptors (like *thin*, *normal* or *soft* in $Q_1$ and $Q_2$) are called "required characteristics," and embody the properties that a record must have to be considered as an element of the answer.

A query also defines the attributes for which required characteristics exist. The set of these input attributes is denoted by $X$. The expected answer is a description over a set of other attributes, denoted by $Y$. Without further precision, $Y$ is the complement of $X$ relatively to $S$: $X \cup Y = S$ and $X \cap Y = \emptyset$.

Hence a query defines not only a set $X$ of input attributes $A_i$ but also, for each attribute $A_i$, the set $C_{A_i}$ of its required characteristics. The set of sets $C_{A_i}$ is denoted by $C$, as shown in the following example.

*Example 1.* Let $Q_1$ and $Q_2$ be the queries stated above. For each query, the sets are:

$Q_1$: $X = \{\text{thickness}\}$, $Y = \{\text{hardness, temperature}\}$,
$C_{\texttt{thick}} = \{\text{thin}\}$ and $C = \{C_{\texttt{thick}}\}$

$Q_2$: $X = \{\text{hardness, temperature}\}$, $Y = \{\text{thickness}\}$,
$C_{\texttt{hard}} = \{\text{soft}\}$, $C_{\texttt{temp}} = \{\text{normal}\}$ and $C = \{C_{\texttt{hard}}, C_{\texttt{temp}}\}$.

When users formulate a question, they expect that all and only the data with the characteristics they specify will be returned. The meaning of that question becomes arguable when many characteristics are expressed for one attribute or when conditions exist for more than one attribute.

The first case is illustrated by the question "how are materials which are malleable or soft?". The question is interpreted in a disjunctive manner as "how are materials with hardness in {malleable, soft}?" and not as "how are materials which hardness is both malleable and soft?". The equivalent query for the correct interpretation is $Q_3$: SELECT thickness, temperature WHERE hardness IN ("malleable", "soft"), interpreted as the condition *hardness = 'malleable' OR hardness = 'soft'*.

The second case is illustrated by the question "how are thick compact materials?". The querying process should put forward only data that comply with the characterization on both thickness and hardness. This precludes, for instance, thick soft materials and thin compact materials from being selected. The equivalent query for this second question is $Q_4$: SELECT temperature WHERE thickness IN ("thick") AND hardness IN ("compact"). The condition of $Q_4$ is interpreted as *thickness = 'thick' AND hardness = 'compact'*.

### 4.2. Evaluation of a query

This section deals with matching one particular summary against a query to decide whether it corresponds to that query and can then be considered as a result. The query is transformed into a logical proposition $P$ used to qualify how one summary relates to the query. $P$ is under a conjunctive form in which all descriptors appear as literals. As a consequence, each set of descriptors yields one corresponding clause.

*Example 2.* The query for "how are the materials which are thin or medium-thickness and moderated or normal-temperature?" is $Q_5$: SELECT hardness WHERE thickness IN ("thin", "medium") AND temperature IN ("moderated", "normal").

In $Q_5$, $X = \{thickness, temperature\}$, $C_{\texttt{thick}} = \{thin, medium\}$ and $C_{\texttt{temp}} = \{moderated, normal\}$. It follows that $P_5 = (thin \vee medium) \wedge (moderated \vee normal)$.

Let $v$ be a valuation function. The boolean value of $P$ depends on each summary $z$: a literal $d$ in $P$ is positively valuated ($v(d) = \texttt{TRUE}$) if and only if $d$ appears in the intensional expression (see Section 3.2) of $z$. Then $v(P(z))$ denotes the valuation of proposition $P$ in the context of $z$.

Let $\mathcal{L}_{A_i}(z)$ be the set of descriptors that appear in $z$ and concern $A_i$. An interpretation of $P$ relatively to query $Q$ leads to discarding summaries that do not satisfy $P$. But, as shown in the following example, some summaries that satisfy $P$ might not match the intended semantics of the query.

*Example 3.* Table 3 shows the characteristics of materials covered by summary $z_2$ from Table 2. Suppose $z_2$ is tested for conformance with a query $Q_6$: SELECT temperature WHERE thickness IN ("thick", "thin") AND hardness IN ("soft", "hard"). $P_6 = (thick \vee thin) \wedge (soft \vee hard)$ and the function $v$ valuates the variables *thin* and *hard* to TRUE as they

**Table 3** Example of descriptor combination

| Candidate | thickness | hardness |
|---|---|---|
| $t_{b_2}$ | thin | soft |
| $t_{c_2}$ | medium | hard |
| $t_{e_1}$ | thin | soft |
| $z_2$ | {thin, medium} | {soft, hard} |

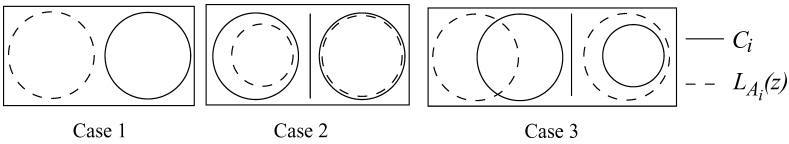


**Fig. 3** Comparison of descriptor sets $\mathcal{L}_{A_i}(z)$ and $C_i$

appear in $z_2$. Consequently, $v(P_6(z_2)) = \texttt{TRUE}$. However, no material that matches $Q_6$ can be found. None of the 3 candidates covered by $z_2$ (see table 3) is an answer of $Q_6$.

This example exhibits the necessity to search leaf nodes of summaries which are supposed to match the query.

While confronting a summary $z$ with a query $Q$, three cases might occur:

- *Case 1*: no correspondence. For (at least) one attribute $A_i$, the set of required labels $C_i$ and the set of the summary labels $\mathcal{L}_{A_i}(z)$ have an empty intersection. Thus $v(P(z)) = \texttt{FALSE}$. In other words, for one attribute or more, $z$ has no required characteristic, i.e., it shows none of the descriptors mentioned in query $Q$.
- *Case 2*: correspondence. The summary being confronted with query $Q$ matches its semantics. It is considered a result if all attributes are one-valued (the summary is a leaf). The following expression holds: $v(P(z)) = \texttt{TRUE}$ and $\forall i,\ \mathcal{L}_{A_i}(z) \subseteq C_i$.
- *Case 3*: no decision can be made. There is (at least) one attribute $A_i$ for which summary $z$ exhibits one or many descriptors besides those strictly required (i.e., those in $C_i$): $\exists i, \mathcal{L}_{A_i}(z) - C_i \neq \emptyset$.
  The presence of required characteristics in each attribute of $z$ suggests, but does not guarantee, that results may be found in the subtree starting from $z$. Exploration of the subtree is necessary to retrieve possible results: for each branch, it will end up in situations categorized by case 1 or case 2. Thus, at worst at leaf level, an exploration leads to accepting or rejecting summaries; the problem of indecision is always solved.

The situations stated above reflect a global view of the matching of a summary with a query. They can also be interpreted, from a crisp set point of view, as a combination of comparisons, still involving $\mathcal{L}_{A_i}(z)$ and $C_i$, concerning one required attribute $A_i$. Figure 3 shows the possible relative positions of $\mathcal{L}_{A_i}(z)$ and $C_i$ and which case they relate to.

### 4.3. Selection algorithm

This section applies the matching procedure from the previous section over the whole set of summaries organized in a hierarchy.

Since the selection should take into account all summaries that correspond to the query, exploration of the hierarchy is complete. The selection (Algorithm 1) is based on a depth-first search and relies on a property of the hierarchy: the generalization step in the SAINTETIQ model guarantees that any descriptor that exists in a node of the tree also exists in each parent node. Conversely, a descriptor is absent from a summary's intension if and only if it is absent from all subnodes of this summary. This property of the hierarchy (which can be easily seen in Table 2) permits branch cutting as soon as it is known that no result will be found. Depending on the query, a part of the hierarchy only is explored. In any case, all relevant results, and only relevant results, are captured.

Algorithm 1 describes the exploration and selection function with the following assumptions:

– function Explore-Select returns a list of matching summaries;
– function Corr corresponds to the matching test reported in Section 4.2;
– operator '+' performs a list concatenation of its arguments;
– function Add is the classical constructor for lists, it adds an element to a list of the suitable type;
– $L_{res}$ is a local variable.

*Example 4.* The result of applying the algorithm on the portion of hierarchy in figure 2 for the queries stated so far is listed below :

| Query | Result list | Tuples |
|---|---|---|
| $Q_1$ | $\langle z_5, z_7 \rangle$ | $t_{b_2}, t_{e_1}$ |
| $Q_2$ | $\langle z_4, z_7 \rangle$ | $t_{d_1}, t_{e_1}$ |
| $Q_3$ | $\langle z_3, z_4, z_5, z_7 \rangle$ | $t_{a_1}, t_{b_1}, t_{c_1}, t_{d_1}, t_{b_2}, t_{e_1}$ |
| $Q_4$ | $\langle\rangle$ | – |
| $Q_5$ | $\langle z_3, z_4, z_5, z_6, z_7 \rangle$ | $t_{a_1}, t_{b_1}, t_{c_1}, t_{d_1}, t_{b_2}, t_{c_2}, t_{e_1}$ |
| $Q_6$ | $\langle z_5, z_7 \rangle$ | $t_{b_2}, t_{e_1}$ |

**Algorithm 1** Function Explore-Select($z$, $Q$)

```
L_res ← ⟨⟩ {the list for this subtree is empty}
if Corr(z, Q) = indecisive then
  for all child node z_child of z do
    L_res ← L_res+ Explore-Select(z_child, Q)
  end for
else
  if Corr(z, Q) = exact then
    {each leaf in the subtree pertains to the answer}
    if z is a leaf node then
      Add(z, L_res)
    else
      {retrieve all the leaves}
      L_res ← L_res+ Explore-Select(z_child, Q)
    end if
  end if{no "else" statement here; it would correspond to case 1}
end if
return L_res
```

### 4.4. Classification

The classification step is an aggregation of selected summaries according to their interpretation with respect to proposition $P$: summaries that have the same required characteristics on all attributes of the input attributes set $X$ form a class that is denoted by $B$.

*Example 5.* In this example, query $Q_5$ from example 2 is employed once again. Since each interpretation of a logical proposition has some boolean variables valuated to true, the set of those variables is considered. The proposition $P_5$ induced by $Q_5$ (see example 2) admits 9 such sets. But the projection of a relation $\mathsf{R}(S)$ on a subset $X$ of its attributes, as done by any

query (see Section 4.1), may cause duplicate values to appear. When such values exist for a query, they are grouped in a class as shown in the following table.

| Interpretation | Relevant summaries | Result (`hardness`) |
|---|---|---|
| {thin, moderated} | $z_5$ | soft |
| {thin, normal} | $z_7$ | soft |
| {medium, moderated} | $z_3, z_6$ | soft, hard |
| {medium, normal} | $z_4$ | soft |

Aggregation of summaries inside a class (for instance, {medium, moderated}) is a union of descriptors: for each attribute $A_i$ of output set $Y$, the querying process supplies a set of descriptors. This set characterizes summaries that respond to the query through the same logical interpretation (i.e., summaries that show the same labels for input attributes).

As a response to a query, the process returns a list of classes along with a characterization of the class for each output attribute. The list is interpreted as follows: *while searching for thin or medium-thickness and normal or moderated-temperature materials, it turned out that:*

– *there exist thin moderated-temperature materials which are soft;*
– *there exist medium-thickness moderated-temperature materials which are either soft or hard;*
– ...

The use of classes has a few benefits:

- the results are expressed in an intensional way;
- one can easily identify which class accounts for an output label;
- it remains possible to provide a single list of output labels by performing a union of labels from all classes.

## 4.5. Experimental results

Figure 4 shows a screenshot of the current implementation of the querying process. The data set used for this specific screenshot describes the images of an image repository based on their color characteristics. For the expression of a query, Larsen (1999) idea that the query

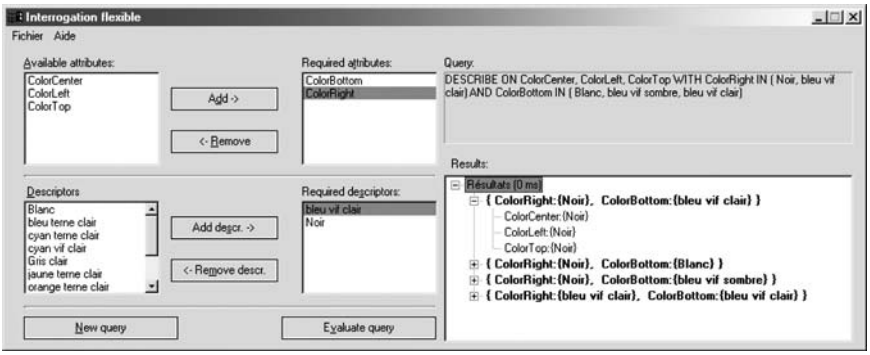


**Fig. 4** Screenshot of the current implementation

**Table 4** Experimental results

| | Dataset A | Dataset B |
|---|---|---|
| Number of summaries | 131 | 27,304 |
| Number of tuples | 74 | 14,269 |
| Summary-based querying | <1 ms | <1 ms |
| Relational DBMS (SQL) | 10 ms–20 ms | 50 ms–60 ms |

language should be "on the human user's condition", that is easy to use, is followed. So, instead of having the users deal with the language, we let them tell which attributes are required and for each of these, which fuzzy labels are wanted. The relevant query is formed by the system and displayed. The list of classes discussed in the previous paragraph can be seen on the right side of Figure 4 with classes in bold face and output characterizations in normal style font.

An experiment has been conducted to compare our prototype with standard SQL. The objective was to roughly evaluate the benefit of using a summary hierarchy. The used procedure has a few steps. First, summaries from the hierarchy are represented into a database. Then, the database is queried using SQL while the equivalent queries are answered through the prototype. Finally, response times are compared for two datasets of different sizes.

Since relational DBMSs (namely Firebird 1.5 and MySQL 4.1 in this test) admit only one-value attributes, the part of a hierarchy that SQL can process is limited to the leaves of the tree whereas the prototype considers the whole hierarchy. The summarized relation R = (thickness , hardness, temperature) is transformed into two tables, SUMMARIES and LABELS, because of normalization. Each summary is transformed into a tuple ⟨ name, thicknessidx, hardnessidx, tempidx ⟩ where name is the primary key and other attributes point to the corresponding labels in table LABELS.

Although two tables are available, only the table SUMMARIES is queried, with all its columns indexed. The results in terms of response time are showed in Table 4. The precision of time measurements is limited by the application programming interface's and operating system's precisions (1 millisecond). The advantage is expected to grow with the size of the dataset but this still has to be proven.

### 4.6. From summaries to tuples in an answer

In the summarization process, the rewriting of data into fuzzy labels produces satisfaction degrees that tell how well a label describes the data. The satisfaction degrees, which are part of a summary's intension, are not used in this paper. But the qualitative information carried by these degrees is interesting, particularly to make a better distinction between results.

Thus, a fully flexible querying process with a ranking of results can be easily built on our approach. Indeed, the summaries hierarchy can be considered as a general index over the data, which can be used to quickly reduce the search space. Thus, a first step (the proposed querying tool) gives answer summaries (leaves of the hierarchy) to a query expressed with linguistic labels. Then a second step retrieves tuples from the summaries extension. Finally, the degrees attached to each tuple can be taken into account to compute a total satisfaction degree of the tuple to the query. This second step is the subject of a future paper.

An additional enhancement would deal with a quantitative aspect. Besides satisfaction degrees, the SAINTETIQ model provides frequency and proportion data attached to descriptors and candidate tuples (Raschia, 2001). By using that data, we will be able to express more

information in a response, for instance the response in example 4 might be "medium-thickness moderated-temperature materials are either soft or hard but only a small part of them is hard".

Thus, taking advantage of the SAINTETIQ summarization process, our tool can perform flexible querying not by applying fuzzy operators on classical data, but by applying boolean inclusion operators on the summaries, i.e., on fuzzy data. This is the main distinctive features of our querying tool, with respect to other flexible querying mechanisms presented in Section 2.

## 5. Conclusion and future research

In this paper, a querying tool for the summarization model SAINTETIQ has been proposed. It allows end-users to efficiently retrieve summaries, and exploits the hierarchical structure of the summaries produced by SAINTETIQ.

From a technical point of view, it performs a boolean matching between the summaries and the query on the basis of linguistic labels from a user-defined vocabulary. It is therefore a classical boolean querying tool whose novelty lies in the use of summaries and in the efficient use of a hierarchy. The querying machinery, as well as a user-friendly interface have been developed and tested on toy examples in order to validate the method.

Then, as it is easy to obtain tuples from summaries, and to rank the tuples according to their membership degree to the summaries, it has been shown that the method can also be considered as a flexible querying tool on a relational database. The flexibility fully relies in the summarization process, and this is one of the reasons why the process is efficient.

Besides, this work is a first attempt at querying the summaries, and the richness of the framework is far from being entirely exploited. Several future developments are under consideration. Among them, the extension of the querying language, adding SQL-like functions as *count()*, the possibility to use a wider vocabulary than the one used in the summarization process, etc.

Above all, expressiveness is the main point future work will focus on as it will eventually allow imprecision in not just the representation of information but also in user queries. It might also cover preferences or priorities in queries as mentioned by Rocacher in Rocacher (2003). An answer from the querying process, for example "thin materials have a soft hardness and a normal temperature", is a description of data. The embedding of the description operation (and others from summary-based querying) as an extension of SQL is a long-term future project.

From the conjunctive normal form expression of queries, determining the reasons of a search failure is simple (see 4.2). From then, an interaction with the user will permit us to implement one of the cooperative behaviors (corrective answers) surveyed by Gaasterland in Gaasterland et al. (1992). The reasons of a failure, that is the fuzzy labels in the query that cause the failure, may be displayed so that the user could ask a new query based on the previous one.

## References

Bosc, Patrick, & Pivert, Olivier (1994). Fuzzy queries and relational databases. In *Proceedings of the ACM Symposium on Applied Computing*, (pp. 170–174). Phoenix, AZ, USA.

Cubero, Juan C., Medina, Juan M., Pons, Olga, & Amparo Vila, Miranda, María (1999). Data summarization in relational databases through fuzzy dependencies. *Information Sciences, 121*(3/4), 233–270.

Dubois, Didier, & Prade, Henri (1997). Using fuzzy sets in database systems: why and how? In *Flexible Query Answering Systems* (pp. 45–59). Kluwer Academic Publishers, Boston.

Dubois, Didier, & Prade, Henri (2000). Fuzzy sets in data summaries—outline of a new approach. In *Proceedings 8th Int. Conf. on Information Processing and Managament of Uncertainty in Knowledge-based Systems (IPMU'2000)*, Vol. 2 (pp. 1035–1040).

Gaasterland, Terry, Godfrey, Parke, & Minker, Jack (1992). An overview of cooperative answering. *Journal of Intelligent Systems, 1*(2), 123–157.

José Galindo, Juan Miguel Medina, Pons, Olga, & Cubero, Juan C (1998). A server for fuzzy SQL queries. In Troels Andreasen, Henning Christiansen, and Henrik Legind Larsen, (Eds.), *Proceedings of the 3rd Int. Conf. on Flexible Query Answering Systems, Roskilde, Denmark*, volume 1495 of *Lecture Notes in Artificial Intelligence LNAI*, (pp. 164–174). Springer.

Kacprzyk, Janusz (1999). Fuzzy logic for linguistic summarization of databases. In *Proceedings of the 8th International Conference on Fuzzy Systems (FUZZ-IEEE'99)*, Vol. 1 (pp. 813–818).

Kacprzyk, Janusz, & Zadrożny, Slawomir (2001). Computing with words in intelligent database querying: standalone and internet-based applications. *Information Sciences, 134,* 71–109.

Larsen, Henrik Legind (1999). An approach to flexible information access systems using soft computing. In *Proceedings of the 32nd Hawaii Int. Conf. on System Sciences*, Vol. 6.

Lee, Do Heon, & Kim, Myoung Ho (1997). Database summarization using fuzzy ISA hierarchies. *IEEE Trans. on Systems, Man and Cybernetics-Part B: Cybernetics, 27,* 68–78.

Raschia, Guillaume (2001). SAINTETIQ*: une approche floue pour la génération de résumés à partir de bases de données relationnelles*. ThËse de doctorat, Université de Nantes.

Raschia, Guillaume, & Mouaddib, Noureddine (2002). SAINTETIQ: A fuzzy set-based approach to database summarization. *Fuzzy Sets and Systems, 129,* 137–162.

Rasmussen, Dan, & Yager, Ronald R. (1997). SummarySQL—A fuzzy tool for data mining. *Intelligent Data Analysis, 1,* 49–58.

Rocacher, Daniel (2003). On fuzzy bags and their application to flexible querying. *Fuzzy Sets and Systems, 140,* 93–110.

Yager, Ronald R. (1988). On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Transactions on Sytems, Man and Cybernetics, 18,* 183–190.

Zadeh, Lotfi A. (1975). The concept of a linguistic variable and its application to approximate reasoning. *Information Sciences, 8,* 199–249 & 301–357. Part III in Vol. 9, pp. 43–80.