Personalized Database Querying Using Data Summaries

Laurent UGHETTO, W. Amenel VOGLOZIN and Noureddine MOUADDIB

***Abstract*— The increasing size of distributed data sources often leads to large amounts of answers to user queries (information overload), and long response times. Moreover, querying systems do not sufficiently take into account and make use of the user-related information (preferences, context, ...) that constitute their profile. In this, they are far apart from personalized systems that adapt their behavior or service to the user. Although user-adaptive systems are mostly found on the Internet, the personalization concept is more general. This paper discusses the possibility and implications of combining the use of summaries with a personalized service, in the querying of a database. The personalization aspect consists in allowing the user to choose their own querying vocabulary, by means of linguistic variables, while the use of summaries increases the efficiency of the querying. Several ways of doing so are identified and search algorithms for the different cases are proposed.**

I. INTRODUCTION

For some years, it has been an undoubted fact that the average size of databases is exploding. The wide availability of computing power and low cost of high-capacity storage units contribute to this phenomenon. So do the interoperability of systems and diversity of data sources. One can safely assume that the amount of data accessible to a user is widely beyond their immediate grasp. Means to circumvent the problem of finding relevant information in large databases include summarization techniques (e.g., see [1], [2], [3], [4]) and, among them, the SAINTETIQ database summarization model [5] which is considered in this paper. Summarization quickens the database querying at a loss of precision, but is of great interest when precision does not matter. Indeed, it leads to condensed, and then more easily comprehensible, answers. However, a limitation relies in the fact that 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. This is why algorithms for querying a database by using summaries of SAINTETIQ have been proposed [6].

On another side, the world wide web revealed user-adaptive systems as an interesting field for researchers and industries. These systems, a.k.a personalized systems, tend to some extent to treat each user as an individual instead of an anonymous service solicitor. Because of the numerous benefits of a user-adapted service [7], personalization systems may have applications even outside the Internet arena (e-commerce, forums, news-feeds, ...).

The authors are with the Laboratoire d'Informatique de Nantes Atlantique – 2 rue de la Houssinière, BP44322, NANTES Cedex 3, FRANCE. (email: laurent.ughetto@univ-nantes.fr, amenel.voglozin@univ-nantes.fr, noureddine.mouaddib@univ-nantes.fr).

This research was partially supported by the French Ministry of Research and New Technologies under the ACI program devoted to Data Masses (ACI-MD), project #MD-33.

Another approach to handle the database size and user-disregarding problems consists in making use of interests, preferences or characteristics of the users. These additional information and constraints allow personalized querying systems to reduce both the search spaces and number of queried sources (thus reducing the response times), and the number of answers (thus reducing the information overload).

This paper tries to mix both approaches and to enlighten the interest of data summaries in the context of personalized querying systems. The studied approach consists in allowing the users to choose their own vocabulary when querying a database, by means of linguistic variables defined in their profile, and then to obtain the required granularity level in the answers. It also consists in making use of database summaries in order to improve the efficiency of the process.

This approach also extends previous works in the field of flexible querying systems for databases, and summaries handling tools (see [8], [9]). Algorithms for searching a database using the predetermined vocabulary of the SAINTETIQ summaries have already been designed [6]. However, they cannot make use of another vocabulary, while most of the flexible querying systems (for instance, SQLf [10] and FQuery [11]) allow the definition and use of query terms on-the-fly. Algorithms proposed here extend the ones in [6] to the use of user-specific vocabulary, by means of linguistic variables stored in their profile.

First, Section II presents the particular kind of database summarization considered in this article, which is based on the introduction of imprecision. It also briefly presents the SAINTETIQ prototype, before discussing the use of summaries in a querying process. Then, Section III shows the possible applications of summaries in personalized systems. Last, Section IV focuses on summaries querying algorithms. Several cases are studied, considering several kinds of personalized vocabularies, and exact or approximate computation. In our opinion, there is no ideal method. The main point of this paper is to discuss different possible approaches only. Thus advantages and drawbacks of each one are pointed out.

II. DATA SUMMARIZATION

This Section briefly introduces the considered kind of data summaries. Then it shows the interest of summaries in querying techniques, and more particularly in the context of personalized access to data.

A. Imprecision-based data summarization

The data summaries considered in this paper are based on the introduction of imprecision. The principle is rather simple. Consider a structured database, whose attributes are defined on "precise" domains, e.g., the set of integers

for attribute `age`. The summarization technique consists in modifying the initial domains into coarser ones. For instance, the domain for `age` is changed from integers (about one hundred values) into a smaller set of values, e.g., {`baby`, `child`, `adolescent`, `young adult`, `adult`, `senior`} containing only 6 values. Then, the database is rewritten using this new vocabulary. As a consequence, numerous tuples become indistinguishable. Indistinguishable tuples are "grouped" into a new one. This new (coarser) tuple, associated with the number of initial tuples it represents *is called a summary*.

Thus, each tuple from the initial database is represented in the summarized database, even though in a coarser way. Other techniques, such as sampling or identifying representative values are not considered here.

The choice of the new attribute domains is a key point of the summarization process. First, these new domains have to be defined according to the user interests, and should be meaningful. They are not designed according to the data to be summarized. Indeed, summarization is not intended here as classification or knowledge discovery tasks. Summaries are produced to outline the database content in a comprehensive way. This point is particularly important in the context of personalized access to data.

It is natural to rewrite continuous or ordered initial domains using intervals (partitioning this domain). Using classical intervals would lead to the well-known threshold effect: a tuple whose value is 17.8 would not be included in a summary whose values are from 18 to 25, although it is very close. Sometimes it is required, for instance when people under 18 cannot vote. Here, the threshold is not artificially added by the summarization process, but exists in the model. In other cases, it is important to avoid this effect, and for that fuzzy linguistic variables [12], which are fuzzy partitions of domains made of overlapping fuzzy sets, can be used.

If making use of fuzzy sets avoid the threshold problem, it leads to another problem. Indeed, as a drawback of the overlapping, a tuple from the initial database can be represented by more than one tuple in the summarized database, as it can belong to several linguistic terms on each attribute. In practice, this is not really a problem, as most often it does not add tuples in the summarized database, but only increases the associated cardinalities.

The granularity of the linguistic variables, representing the new domains of attributes, determines both the ratio (number of summaries w.r.t number of initial tuples) and precision of the summaries. Indeed, the coarser the new domain, the more compact the obtained database summary, but also the more imprecise the representation. If it is too fine, the data is hardly summarized. If it is too coarse, the summaries become not informative enough.

The granularity level should also be defined according to the users needs. Too fine, and the user will get too many and too precise answers. Too coarse and the user will get too imprecise, and potentially useless answers.

This is why it is interesting to use several levels of granularity when summarizing a database. These levels could be obtained either by using thesauri on each attribute, or by grouping terms of the linguistic variables in a hierarchical way.

B. The SAINTETIQ summarization model

These features are implemented in the structured data summarization model SAINTETIQ developed by our research team ([5], [13]). SAINTETIQ is founded on an incremental classification algorithm, and produces linguistic summaries structured in a hierarchy. The hierarchy building process, consisting of two steps, is explained on a simple example.

Consider the relation R = (thickness, hardness, temperature) from a `MATERIALS` table. A tuple from R describes a material used in an imaginary metallurgy plant to produce square sheets. Attribute `thickness` is expressed in mm and has a limited range (from 0.15 to 50). Attribute `hardness` is the final product's expected value on scale B of the Rockwell hardness test. Attribute `temperature` is a material's melting point.

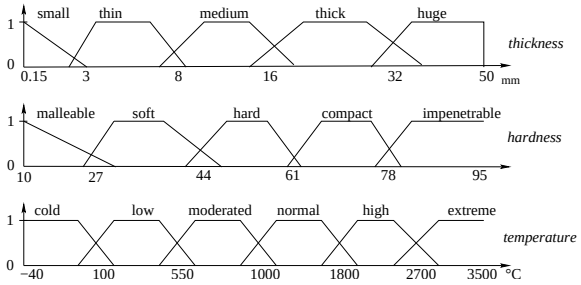

Fig. 1. *Linguistic variables for the `MATERIALS` table*

The linguistic variables associated with the attributes of R are shown on Fig. 1. Consider the `MATERIALS` table composed of the 5 tuples: $t_a = \langle 10, 38, 900 \rangle$ (copper and zinc alloy CuZn40), $t_b = \langle 8, 40, 850 \rangle$ (copper and tin alloy CuSn12), $t_c = \langle 12, 44, 896 \rangle$ (copper and arsenic alloy CuAs05), $t_d = \langle 10, 35, 1530 \rangle$ (iron Fe) and $t_e = \langle 5, 35, 1453 \rangle$ (nickel Ni). Rewriting these 5 tuples using the linguistic variables leads to the candidate tuples shown on Fig. 2. The generated summaries that appear in the hierarchy on Fig. 3 are described on Fig. 4.

Material	Candidate tuples
UZ40	$t_{a_1} = \langle$ 0.7/medium, 1.0/soft, 0.85/moderated $\rangle$
CuSn12	$t_{b_1} = \langle$ 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	$t_{c_1} = \langle$ 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	$t_{d_1} = \langle$ 0.7/medium, 1.0/soft, 0.85/normal $\rangle$
Ni	$t_{e_1} = \langle$ 1.0/thin, 1.0/soft, 0.96/normal $\rangle$

Fig. 2. *Translation of tuples from table `MATERIALS`*

In the first step, the tuples are translated using the linguistic variables (which are part of a background knowledge provided by the user). Each attribute value of a tuple is

rewritten using one of the corresponding linguistic labels. The rewritten tuples are called *candidate tuple*. As one attribute value may correspond to more than one fuzzy label (e.g., *8 mm* is described by *medium* and *thin*), one tuple (for instance t_b and t_c on Fig. 2) may yield many candidate tuples.

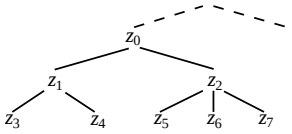

Fig. 3. *Part of the summary hierarchy for* `MATERIALS`

In the second step, each candidate tuple is incorporated into a tree and reaches a leaf node. This can be seen as a classification of the candidate tuple. The building process is not detailed here, for the sake of brevity (see [5] for details). It is sufficient here to notice that the process is incremental. It then suffers from an order effect (the structure of the tree depends on the order the tuples are incorporated in). To limit this order effect, the tree is modified throughout candidate tuples incorporation, by means of learning operators.

Summary	Intension
z_3	⟨ 1.0/medium, 1.0/soft, 1.0/moderated ⟩
z_4	⟨ 0.7/medium, 1.0/soft, 0.85/normal ⟩
z_5	⟨ 0.35/thin, 0.9/soft, 1.0/moderated ⟩
z_6	⟨ 1.0/medium, 0.4/hard, 0.9/moderated ⟩
z_7	⟨ 1.0/thin, 1.0/soft, 0.96/normal ⟩
z_1	⟨ 1.0/medium, 1.0/soft, 1.0/moderated + 0.85/normal ⟩
z_2	⟨ 1.0/thin + 1.0/medium, 1.0/soft + 0.4/hard, 1.0/moderated + 0.96/normal ⟩
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 ⟩

Fig. 4. *Description of some summaries*

In the hierarchy structure, a level can be associated with the relative proportion of data that is described by a summary: the deeper the summary in the tree (or the lower its level in the hierarchy), the finer the granularity.

Thus the lowest level (the leaves) contains the most precise and specific summaries. Such summaries are similar to candidate tuples (i.e., the rewritten tuples) in their intensional expression ($z = \langle \alpha_1/d_1, \alpha_2/d_2, \ldots, \alpha_n/d_n, \rangle$): there is only one label per attribute in these leaf-summaries (see z_3 on Fig. 4).

Then, each intermediate node is just the grouping of its children nodes, and then is described with at least one multi-labeled attribute (e.g. z_1 and z_2). These labels are the union of the ones in the children summaries.

As a consequence, the root of the tree is the most general summary, and covers all the data.

Another convenient feature in SAINTETIQ consists in embedding references of the represented initial tuples in the leaf-summaries. When precise answers are needed, it is then possible to directly retrieve initial tuples from summaries.

C. Why querying 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 sub-queries are expressed. Waiting for an answer for a long time is frustrating, particularly when the query fails.

Database summaries are a means of significantly reducing the volume of input for processes that require 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 [5].

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. For instance, when trying to anticipate bird flu spread, one needs to estimate "how many" "big" poultry farming are "near" some flight path of migratory birds. The precise location, exact number of birds... is useless, and interfering information. The user can then choose the adequate granularity level in both his request and the answers.

Sometimes, imprecision can be a requirement. This is 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. This is the case when one wants to know if a database is likely to answer the query. This point is of interest in the context of distributed information sources. One can first query the database summaries in order to rapidly determine if the database can contain a relevant answer. Then, the precise query, with much longer response times, is sent to a limited number of well-chosen sources [14].

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. The use of summaries can even be transparent to the user.

This is also of interest in the context of flexible querying. In particular when the queries and the summarization process use the same linguistic terms, efficient querying algorithms can be used to get the answer summaries [8], [9]. Then, precise tuples can be retrieved from these summaries. In this scenario, the hierarchical tree of summaries acts as a multidimensional index.

III. QUERYING SUMMARIES IN THE CONTEXT OF PERSONALIZATION

A. Summaries and personalized access to data

In the previous section, is has been shown that querying summaries is interesting, especially when the vocabulary

used to write queries, and/or to express results is coarser than the one used to describe the tuples in the database. This is clearly a kind of flexible query answering system.

Furthermore, as the vocabulary used is different from the one defined in the database (the domains of attributes), it is not necessarily set by the DBMS. In fact, it should be defined according to the user. This is where personalization is involved.

Indeed, consider a user querying a housing agency database for "cheap" apartments. What is meant by "cheap" can vary from one user to the other (e.g., it depends on the user income), or from one kind of user to the other (e.g., "cheap" does not have the same meaning for buyers and for sellers). This why it is not suitable to define such linguistic terms in the DBMS, not taking the users into account.

As a first and direct idea, linguistic variables, which give a mapping from linguistic terms (the new vocabulary) to precise domain values (see Fig. 1), can be part of the users' profile. Each user defines his or her own vocabulary, to be used by the DBMS.

Furthermore, a summary of the database, created with the very same vocabulary is added to the profile. This idea, along with its limitations, is studied in the next section.

B. Querying a personalized database summary

In this section, it is assumed that a user has defined, by means of linguistic variables, a vocabulary he wants to use when querying a huge database.

First, summaries are not considered. When a query is sent by the user, using his own vocabulary, the personalization systems has to rewrite this query, to access the DBMS to get answers, and then to rewrite the answers according to the user's vocabulary (i.e., the user profile).

Although the answer is satisfactory, this querying method is unnecessarily time consuming, especially in case of big databases, where response times can become prohibitive. This is partly due to the fact that the first rewriting and the database access involve a precision level which is too fine with respect to the one required in the answer.

In order to decrease this response time, and avoid the unnecessarily precision in the querying mechanism, one can make use of SAINTETIQ-like summaries.

Now, suppose that a database summary is built using the linguistic variables defined by the user, and stored in his profile.

In this case, querying algorithms have been proposed [6], [9]. These algorithms are particularly efficient since they involve boolean operations only (while manipulating fuzzy sets), and take advantage of the hierarchical structure of the SAINTETIQ summaries, which acts as a multidimensional index. These algorithms have been implemented, and experiments have shown a noticeable response time gain w.r.t. "classical" querying (the efficiency of the SAINTETIQ summaries hierarchy w.r.t. other multidimensional indexes is still under investigation).

This approach presents several advantages, but also some important drawbacks. Among the advantages, one finds the possibility for a user to define his own querying vocabulary, and to get very quick answers, expressed with the right precision level, from a database summarized according to this vocabulary. But this cannot be obtained at no cost.

A first drawback is due to the necessity to summarize the database using the linguistic variables from the user profile. The number of summaries to be created and stored amounts to the number of users. If there is a limited number of users, the summaries can be stored by the DBMS. If the summaries are not too big (i.e., they summarize a middle-sized database), storing them in the user's profile can be considered. In case of a huge database and numerous users (which is the most likely case), specific and acceptable storage mechanisms remain to be found.

Another drawback is due to the summarization time. Even if the process is incremental, with polynomial complexity, it is time consuming. From a computational complexity point of view, the process then suffers from the same kind of problem as explained above from a spatial complexity point of view. Moreover, from the user point of view, summarization has to be anticipated. From a system point of view, using *personalized* summaries is only conceivable when numerous queries are to be sent to the database from the same user, or when response time is very important, and is worth the anticipated computation time.

Clearly, on a large scale, the ideal solution (from the querying point of view) presented in this section is hardly conceivable.

In the next sections, two solutions to avoid these complexity problems are proposed. The first consists in using group profiles in order to reduce the number of created and updates summaries. The second is more drastic. Only one database summary is created, and updated by the DBMS, using predetermined linguistic variables, while the queries are still expressed with the user profiles.

C. Summarization according to group profiles

As the database size, and the summaries sizes cannot be really decreased, the only way to decrease the above-mentioned complexity is to reduce the number of summaries. The real number of users cannot be decreased either. Yet, each user can no more be allowed to define his own linguistic variable.

The idea here is to use group profiles. A limited number of user classes are defined, with well-adapted linguistic variables to each class. Each user class represents a *group of users* who share the same profile. The DBMS has to create and update the corresponding summaries. And each user can "subscribe" to one group of users whose profile they share.

This number of groups is determined according to both the capabilities of the DBMS, and the particular application. Consider for instance a housing agency database. As said before, "cheap" has not the same meaning for everybody, as well as "large", or "well-located". On a small system, two main groups have to be created, one for sellers, and the other for buyers. On bigger systems, other groups can

be created, as “personal buyer”, “investor”, “personal seller” which take into account the kind of buyer or seller. More detailed categories can be imagined, depending on annual income, profession, age, etc.

This solution has several advantages. First, by contrast with the solution in the previous section, the number of summaries to create and update is limited and controlled. Then, as the querying vocabulary remains the one used to create the summaries, the above-mentioned efficient querying algorithms can be used.

The main drawback of this approach is that users cannot use their own vocabulary. They have to subscribe to a group profile, and then share the group’s linguistic variables. That means the user has to *learn* the linguistic terms of his group before using them accurately.

At worst, this is not easier than using ad hoc linguistic terms, predetermined in a DBMS. Indeed, one goal of personalization is to define in the user’s profile the exact meaning of the linguistic terms they will use in queries. The user can “speak his own language”, without having to learn someone else’s.

This solution is acceptable from a system point of view, but far from ideal from a personalization point of view.

IV. Querying summaries using a personalized vocabulary

In this section, only one summary of the database is considered. It is created using a priori defined linguistic variables. Thus, the summarization process, as well as the update process can be managed by the DBMS.

Nevertheless, the user defines his own linguistic variables in his profile, and will use them when querying the summarized database. As the vocabulary in the queries is different from the one in the summaries, the algorithms previously proposed (see [6], [8]) cannot be used directly.

This section is dedicated to the study of summaries querying algorithms when the querying vocabulary is different from the one used to create the summaries.

A. Precise queries

First consider the case of “precise” queries, i.e., queries expressed with precise values. These precise values are the ones from the initial attribute domains, on which the linguistic variables are defined.

For instance, in the metallurgy plant example detailed above, a query with precise values can be “what is the hardness of materials whose melting point is 1530”.

The main point is that summaries cannot give more precise information than the one they contain.

As a consequence, the precise temperature value 1530 has to be rewritten as a linguistic term. Here, 1530 is turned into “normal”, according to the linguistic variable shown on Fig. 1.

Once the query is rewritten, the boolean summaries querying algorithm can be applied. Suppose the system answers “soft or hard” to the rewritten query: “what is the hardness of materials whose melting point is *normal*”.

On a first step, the answer which can be given is nothing more than: “materials whose melting point is normal are soft or hard”. According to the semantics of SaintEtiQ summaries, this answer means that in the database, there are materials whose couple (hardness, temperature) is (“soft”, “normal”) or (“soft”, “hard”). The number of such materials, as being part of the summaries, can be given as well. However, it gives no information on material whose melting point is exactly 1530. This is not surprising, as summarization is based here on the introduction of imprecision.

On a second step, in case a precise answer is requested, one can retrieve the tuples represented by the summaries (“soft”, “normal”) and (“soft”, “hard”), as record IDs (i.e., references to tuples) are included in the summaries representation. This restricted set of tuples has just to be filtered with the initial criterion (melting point = 1530).

However, in that case where precision is requested, the use of summaries is questionable. If the summaries exist, querying them first presents two main advantages. First, in case of null answers (e.g., no “normal” fusion temperature materials used in the factory), the null answer is determined very rapidly, and tuples close to the requested ones can even be proposed (see [6]). Second, in the other cases, the summaries hierarchy used in the algorithm acts as a multidimensional index which will quicken the search. The efficiency of the summaries as multidimensional index is still under evaluation, but seems promising.

This leads to Algorithm 1. In this algorithm, information such as “precise value required” or “repair in case of null answers” can be part of the user’s profile, then avoiding interaction.

Algorithm 1 Querying summaries with a “precise” query

```
Input: QueryInit // Initial query from the user
Input: SummTree // Tree of summaries produces by SaintEtiQ
Input: Profile // Profile of the user
  // Rewrite the query, according to the user linguistic variables
  Q ← Rewrite(QueryInit, Profile.LinguisticVar)
  // Query the Tree of Summaries using algorithms in [8]
  AnsSet ← SummQueryingAlg(SummTree, Q)
  // Dealing with null answers
  if AnsSet == {} AND Profile.RepairNullQuery == yes then
    // Query the Tree of Summaries using repair alg. in [6]
    AnsSet ← SummQueryingRepair(SummTree, Q)
  end if
  // Dealing with imprecision
  if Profile.Precision == yes then
    // Retrieve precise tuples from the answer summaries
    PrecAnsSet ← {}
    for all S in AnsSet do
      PrecAnsSet ← PrecAnsSet ∪ TuplesFromSumm(S)
    end for
    // Filter out the tuples according to initial query
    FinalPrecAnsSet ← FilterOut(PrecAnsSet, QueryInit)
  end if
Output: AnsSet // set of answer summaries (maybe empty)
Output: FinalPrecAnsSet // set of answer tuples (if required only)
```

Note that a precise value can be rewritten in several ways, when it belongs to more than one linguistic term. In this

case, it is sufficient to rewrite the precise value with one term only (any of them). This particular behavior comes from the summaries building algorithm. Indeed, before a tuple from the initial database is inserted in the summarization tree, it is rewritten using the linguistic variables. When, for some attribute, it belongs to more than one linguistic term, it is rewritten several times, with each linguistic term it belongs to. As a consequence, this tuple is represented in (and referenced by) several summaries.

In brief, when precise (classical) queries are considered, the SAINTETIQ summaries hierarchy is used either directly or as a multidimensional index. Directly when imprecise answers (expressed with the summaries linguistic variables) are acceptable, after just rewriting the query. As a multidimensional index when precise answers are requested. Indeed, in this case, the summaries give a superset of the answer set, which has to be filtered according to the request.

B. Imprecise queries - Exact computation

Now consider imprecise requests, i.e., queries expressed using linguistic terms defined in the users' profile. Depending on the users' preference, the answers can be precise (tuples from the initial database), imprecise defined with the summaries linguistic variables, or imprecise defined with the users' linguistic variables. However, the second case is questionable. Indeed, if users do not accept the linguistic variables of the summaries in the queries, why would they accept them in the answer? Thus, this case will not be considered.

This section shows how Algorithm 1 can be adapted to the case of imprecise queries, leading to an exact computation of the results. Then, approximate (and faster) algorithms are proposed in the next section.

Basically, the precision of queries is not really a characteristic of Algorithm 1. Then, the main part of this algorithm is the same: only the query rewriting and the filtering parts are modified.

As previously, the algorithm consists in obtaining the smallest set of summaries which represent all the answer tuples. Then, once the tuples are retrieved from the summaries, the filtering step consists in rejecting false positives. There are no false-negative tuples. And finally, depending on the user's choice, the retrieved tuples can be rewritten using their linguistic variables. In fact, this last step consists in "summarizing" the retrieved tuples, using the user's linguistic variables.

In order to obtain the smallest set of summaries which represent all the answer tuples, the query has to be rewritten. For each attribute, the linguistic terms of the query (from the user's linguistic variables) are rewritten by the smallest superset of linguistic terms from the summaries linguistic variables, using the fuzzy inclusion:

$$F \subseteq G \quad \text{iff} \quad \forall x \in F,\ \mu_F(x) \leq \mu_G(x)\ .$$

An example of such a rewriting is shown on Fig. 5.

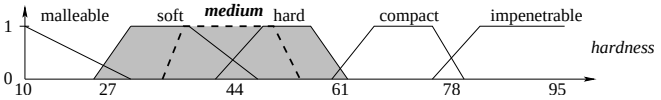

Fig. 5. *The user's term **medium** (dashed line) is rewritten with the terms of the summaries "soft or hard" (depicted in gray).*

Querying the summaries and retrieving the answer tuples is performed as in Algorithm 1.

Then, if the users require precise answers, the retrieved tuples are just given. If they want answers expressed with their linguistic variables, the tuples have to be rewritten accordingly. As this rewriting process is a summarization process, SAINTETIQ could be used for this task, for instance in its web service form described in [13]. For instance, this could allow the user to get the most informative granularity in the representation of the answer.

If only few tuples are considered, a simpler results-formatting algorithm can be considered. It roughly consists in rewriting each tuple with the accurate linguistic term (and keeping only one occurrence of similar rewritings), as on the one-dimensional example of Fig. 6. This is equivalent to considering only leaf-summaries in SAINTETIQ.

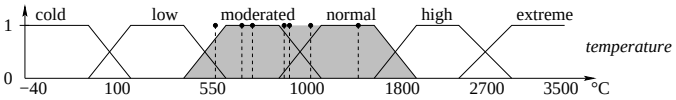

Fig. 6. *The 7 answer values are "summarized" with the user's terms "moderated or normal" (depicted in gray).*

This procedure can be roughly described by Algorithm 2 (repairing queries in case of null answer has been removed for the sake of simplicity).

Algorithm 2 exact querying algorithm with users' linguistic terms

Input: QueryInit // *Initial query from the user*
Input: SummTree // *Tree of summaries produces by SAINTETIQ*
Input: Profile // *Profile of the user (containing their linguistic terms)*
// *Rewrite the query with the user's linguistic variables as in Fig. 5*
Q ← Rewrite(QueryInit, Profile.LinguisticVar)
// *Query the Tree of Summaries using algorithms in [8]*
AnsSet ← SummQueryingAlg(SummTree, Q)
// *Retrieve precise tuples from the answer summaries*
PrecAnsSet ← {}
for all S in AnsSet **do**
  PrecAnsSet ← PrecAnsSet ∪ TuplesFromSumm(S)
end for
// *Filter out the tuples according to initial query*
PrecAnsSet ← FilterOut(PrecAnsSet, QueryInit)
// *Rewrite (summarize) answer tuples with user's terms*
FinalAnsSet ← Summarize(PrecAnsSet)
Output: FinalAnsSet // *set of answer expressed with user's terms*

The main drawback of this method is the computational time, as it comes from the summaries terms to the users' terms through the database tuples. Now, the difference between this procedure and a personalized classical querying

system is the use of the database summaries as a multidimensional index to retrieve the tuples. Indeed, a system which would make use of the user's linguistic terms (in both the queries and the answers) and a classical querying system would have the same rewriting processes, both for the queries and the answers formatting. Thus, retrieving the tuples from the database is not a step due to the use of summaries, but inherent to the personalization process. Among the advantages of the summaries, along with the above-mentioned use as a multidimensional index, the SAINTETIQ summarization process can be used to format the answers according to the users' profile.

Even if retrieving the database tuples could not be avoided for an *exact* computation of the results, it would be interesting to *jump* straightforwardly from the summaries linguistic terms to the users' linguistic terms. This is the topic of the next section.

C. Imprecise queries - Approximate computation

When a short response time is preferred to an exact answer, one can avoid retrieving the database tuples in the algorithm from the previous section. This can be achieved in two different ways, with two different meanings of the results.

First consider Algorithm 3. It is obtained straightforwardly from Algorithm 2. The query is rewritten, and the summaries queried the same way. Once the answer summaries are retrieved, they are now directly rewritten using the user's linguistic variables.

The answer summaries (`AnsSet`), to be rewritten using the user's vocabulary, represent a set containing all the answer tuples. The only way to rewrite the answer summaries without losing any of the answer tuples is to build a superset of the summaries.

Algorithm 3 Approximate, fast querying algorithm

Input: QueryInit // *Initial query from the user*
Input: SummTree // *Tree of summaries produces by SAINTETIQ*
Input: Profile // *Profile of the user (containing their linguistic terms)*
// *Rewrite the query with the user's linguistic variables as on Fig. 5*
Q ← Rewrite(QueryInit, Profile.LinguisticVar)
// *Query the Tree of Summaries using algorithms in [8]*
AnsSet ← SummQueryingAlg(SummTree, Q)
// *Rewrite answer summaries with user's terms*
FinalAnsSet ← Rewrite(AnsSet, Profile.LinguisticVar)
Output: FinalAnsSet // *set of answer expressed with user's terms*

The final answers are similar to sets of summaries. For the users to accurately make use of these answers, their meaning has to be clearly understood.

An empty set of answers (`FinalAnsSet`) guarantees that there is really no answer tuple. A non-empty set only means that *it is possible that* answer tuples exist, but nothing can be guaranteed. This is due to the first rewriting, as the rewritten query captures more tuples than the initial one. However, if there exist *real* answer tuples (the ones retrieved by Algorithm 2), they are covered by the set of answers (`FinalAnsSet`).

The advantages of Algorithm 3, are the efficiency of the computation and the use of the user's vocabulary. Its drawback, besides a new loss of precision, is the fact that only null answers can be guaranteed.

In some cases, it can be more interesting to guarantee that there are answers instead of guaranteeing null answers only. This can be achieved using another algorithm, similar to Algorithm 3, but for the way rewritings are done.

So far, the initial query has been rewritten such that "nothing is lost" (false positive are added, but no false negative are forgotten). By contrast, in order to guarantee the obtained query, no false positive should be added. Then, the initial query has to be rewritten using subsets of linguistic terms, as shown on Fig. 7.

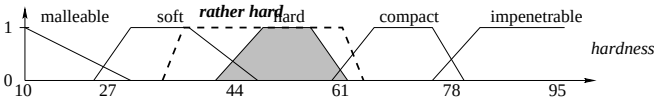

Fig. 7. *The user's term **rather hard** (dashed line) is rewritten with the terms of the summaries "hard" (depicted in gray).*

The second rewriting is achieved the same way: the final answer is a subset of answer summaries, expressed with the user's linguistic terms. In this case, the final answer has the following meaning. In case of a null answer, nothing can be guaranteed. Indeed, if it guarantees that there are no false positive tuples in the final answer, the rewriting processes may discard false negative tuples. Thus, when the algorithm gives a null answer, it can be either because there is no answer tuple or because all answer tuples have been discarded. In case of a non-null answer, answer tuples (from the initial database) exist for sure and are located in the given answer. However, not all the answer tuples are captured in this case.

The drawback of this algorithm is that a null answer is likely to be obtained when the granularity in the query and in the database summaries is comparable. Indeed, non-null answer can be obtained only if linguistic terms from the summaries are included in the ones from the query, for each attribute. That means only if the query if "very" imprecise.

If this algorithm has to be used, it can be wise to choose a fine partition of the linguistic variables domains for the database summarization, with the drawback of a less efficient summarization process.

Also note that the two approximate algorithms proposed here can be used together, as one guarantees null answers and the other guarantees non-null answers. This dual approach can be viewed as reasoning with both possibility and necessity (see e.g. [15]).

V. CONCLUDING REMARKS

In this paper, the use of data summaries in personalization of database querying has been studied. This work can be seen under two aspects, first as introducing summarization

techniques into a flexible query answering system, and second as introducing personalization (by means of user-chosen linguistic terms) in a querying system based on summaries.

Under the first aspect, the main advantages of using SAINTETIQ summaries is a response time gain, either because the summaries act as a multidimensional index, or because the summaries only are queried, instead of the whole database. This is of interest when huge databases are considered. About the specific aspect of indexing, this work has shown close links between the considered kind of summarization process and multidimensional indexing. These links are currently studied.

About the second aspect, it has been shown that the use of linguistic terms allows the users to define their own meaningful vocabulary, and then the most appropriate granularity in both the query and the answer.

Several methods have been proposed. From a querying point of view, the best method consists in building user-specific summaries, as in this case the queries and summaries share the same vocabulary. As a consequence, the querying algorithms are based on boolean (and then efficient) operations. But maintaining user-specific summaries is hardly conceivable when numerous users are considered.

Then, two alternatives have been proposed. The first one consider group profiles in order to reduce the number of managed summaries. In this case, the querying process remains boolean, but the users have to use the vocabulary of their group. This is not ideal from a personalization point of view. The second alternative consists in building only one SAINTETIQ summaries hierarchy using ad hoc linguistic terms, and querying these summaries with other linguistic terms. Several querying algorithms have been proposed in this case, allowing for exact, or approximate computation.

However, this work was a first step in the querying of summaries with ad hoc linguistic terms. In this first step, only simple conditions have been considered, which lead to several limitations, in particular from a personalization point of view. For instance, the users' vocabulary, stored in their profile has been limited to one linguistic variable per attribute. This limitation does not allow the user to adapt the granularity level from one query to the other, or to ask for different granularity levels in the query and the answer.

Then a next step will consist in extending the users' profile management.

In addition, it has been shown that the granularity of the linguistic variables used to build the database summary is of importance for both the summarization process, and the accuracy of the approximate algorithms answers. The consequences of this granularity remain to be precisely studied, for the right level to be automatically defined when summarizing a database.

REFERENCES

[1] J. C. Cubero, J. M. Medina, O. Pons, and M. A. V. Miranda, "Data summarization in relational databases through fuzzy dependencies," *Information Sciences*, vol. 121, no. 3–4, pp. 233–270, 1999.

[2] D. Dubois and H. Prade, "Fuzzy sets in data summaries — outline of a new approach," in *Proceedings 8th International Conference on Information Processing and Managament of Uncertainty in Knowledge-based Systems (IPMU'2000)*, vol. 2, jul. 2000, pp. 1035–1040.

[3] J. Kacprzyk, "Fuzzy logic for linguistic summarization of databases," in *Proceedings of the 8th International Conference on Fuzzy Systems (FUZZ-IEEE'99)*, vol. 1, aug. 1999, pp. 813–818.

[4] D. H. Lee and M. H. Kim, "Database summarization using fuzzy ISA hierarchies," *IEEE Trans. Syst., Man, Cybern. B*, vol. 27, pp. 68–78, feb. 1997.

[5] G. Raschia and N. Mouaddib, "SAINTETIQ: a fuzzy set-based approach to database summarization," *Fuzzy Sets and Systems*, vol. 129, pp. 137–162, 2002.

[6] W. A. Voglozin, G. Raschia, L. Ughetto, and N. Mouaddib, "Querying the SaintEtiQ summaries – dealing with null answers," in *Proceedings of the 14th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE'05)*, Reno, Nevada, may 2005, pp. 585–590.

[7] J. Fink and A. Kobsa, "A review and analysis of commercial user modeling servers for personalization on the world wide web," *User Modeling and User-Adapted Interaction*, vol. 10, no. 2–3, pp. 209–249, 2000.

[8] W. A. Voglozin, G. Raschia, L. Ughetto, and N. Mouaddib, "Querying the SAINTETIQ summaries – a first attempt," in *Proceedings of the 6th International Conference on Flexible Query Answering Systems (FQAS'04)*. Springer, Lyon, France, jun. 2004, pp. 404–417.

[9] ——, "Querying a summary of database," *International Journal of Intelligent Information Systems (JIIS)*, accepted 2005, to appear.

[10] P. Bosc and O. Pivert, "Fuzzy queries and relational databases," in *Proceedings of the ACM Symposium on Applied Computing*, Phoenix, AZ, USA, 1994, pp. 170–174.

[11] J. Kacprzyk and S. Zadrożny, "Computing with words in intelligent database querying: standalone and Internet-based applications," *Information Sciences*, vol. 134, pp. 71–109, may 2001.

[12] L. Zadeh, "The concept of a linguistic variable and its application to approximate reasoning," *Information Sciences*, 1975, part 1: 8:199-249; Part 2: 8:301-357; Part 3: 9:43-80.

[13] R. Saint-Paul, G. Raschia, and N. Mouaddib, "General purpose database summarization," in *Proceedings of the 31st VLDB Conference*, Trondheim, Norway, aug.–sept. 2005, pp. 733–744.

[14] B. Benatallah, M. Hassan, N. Mouaddib, H. Paik, and F. Toumani, "Building and querying an e-catalog network using p2p and data summarisation techniques," *International Journal of Intelligent Information Systems (JIIS)*, accepted 2004, to appear.

[15] D. Dubois, H. Prade, and L. Ughetto, "A new perspective on reasoning with fuzzy rules," *International Journal of Intelligent Systems*, vol. 18, no. 5, pp. 541–563, 2003.