A survey of query recommendation techniques for datawarehouse exploration

Patrick Marcel*, Elsa Negre**

*Université François Rabelais Tours, Laboratoire d'Informatique, France
patrick.marcel@univ-tours.fr,
** LINA (UMR CNRS 6241) - Nantes University, France
elsa.negre@univ-nantes.fr

Abstract. Lots of data are gathered in datawarehouses that are navigated and explored for analytical purposes. Only recently has the problem of recommending a datawarehouse query to a datawarehouse user attracted attention. In this paper, we propose a simple formal framework for expressing datawarehouse query recommendations. We propose to see the problem of recommending a datawarehouse query for exploration purposes as a function computing a set of queries and associated ratings given a query log, a session, a user profile, a datawarehouse instance, and an expectation function. The rating computed indicates the usefulness of each query for a session. With this viewpoint, we review and categorize the few techniques that, to the best of our knowledge, have been proposed and we illustrate them with a case study.

1 Introduction

Lots of data are gathered and shared in databases that are navigated and explored for analytical purposes. Only recently has the problem of recommending a database query to a database user attracted attention (Chatzopoulou et al., 2009; Khoussainova et al., 2009; Stefanidis et al., 2009). However, in other contexts (like e.g., e-commerce or web search) the problem of computing recommendations is deeply investigated (Adomavicius and Tuzhilin, 2005; Baeza-Yates, 2010).

A typical example of database analysis is a datawarehouse navigated by decision makers using OLAP queries (Sarawagi, 2000). A datawarehouse can be seen as a large database with a particular topology, shared by many analysts who have various interest and viewpoints, explored by sequences of queries. In such a context, query recommendation is particularly relevant since the user is left with the tedious task of navigating a large datacube to find valuable insight.

In this paper, we survey the existing methods for computing datawarehouse query recommendations. We restrict the scope of this survey to methods that, given a user's query over a datawarehouse, use it or transform it into another query, with a supposed added value for the user's exploration. We propose a formal definition of this problem, namely to see the recommendation of datawarehouse queries for exploration purposes as a recommending function taking as input: The datawarehouse query log, a particular query session called the current session, a user profile, a datawarehouse instance, and an expectation function. Given these parameters this recommending function outputs a set of recommended queries, each with a given rating indicating the interest of the query for the current session. We motivate this viewpoint by reviewing the peculiarities of datawarehouses exploration, and we position this viewpoint w.r.t. the ones given in Stefanidis et al. (2009) and Golfarelli (2010). With this viewpoint, we review and categorize into four different approaches the few methods that, to the best of our knowledge, exist. These approaches are then illustrated with a case study on a toy example of OLAP exploration.

Note that we do not claim that our viewpoint allows to describe every possible recommendation techniques. Instead our goal is to use it as a formal framework for describing the existing approaches that suggest queries to further explore a datawarehouse during an analytical session.

This paper is organized as follows: Section 2 exposes the problem of datawarehouse query recommendation with an intuitive example. Section 3 recalls the classical viewpoint on recommendation and the existing approaches in databases. Section 4 introduces our point of view on query recommendation in datawarehouse, and Section 5 introduces and categorizes the few works that propose a method for helping the user to explore a datawarehouse by suggesting queries. Section 6 is a case study illustrating the different approaches in the context of a user navigating a multidimensional database for on-line analytical purposes. Finally, Section 7 concludes and proposes research directions.

2 Motivation

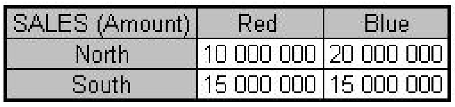
In this section, we illustrate with an informal example various methods for recommending queries for datawarehouse exploration purposes. The context of this example is that of a user navigating a datawarehouse. In our example, the datawarehouse records sales of Vehicles in different Locations at different Times. These sales are recorded as tuples called fact, an example of which would be $(red, North, 2009, amount, 10.000.000)$ that indicates that the sales amount of red vehicles in region North in 2009 is 10.000.000.

Consider a user, called the current user, who launched a sequence of queries $s_c = \langle q_1, q_2 \rangle$, called the current session, where q_1 is the first query launched, and q_2 the second one (and the last one, called the current query, the result of which is depicted Figure 1). Suppose these queries are respectively:

1. *Sales of all vehicles in France*
2. *Sales of red or blue vehicles in the regions North or South*

The problem we focus on is: What would the next query be to pursue the navigation? Various proposals are possible.

First, it could be recommended a query on values that relate to the user's profile. Suppose that it is known that the user is interested in 'Paris' if she queries data concerning region North and that she prefers last year's data. A possible recommendation could be to modify her current query to incorporate these values, namely: *Sales of red or blue vehicles in region 'North', 'South' and city 'Paris' in '2009'.*



SALES (Amount)	Red	Blue
North	10 000 000	20 000 000
South	15 000 000	15 000 000

FIG. 1 – *Result of the current query*

Second, it could be recommended a query whose result shows something surprising w.r.t. users' expectations. Let us have a closer look at the current query result of Figure 1.

It indicates that the sales of blue vehicles are twice the sales of red vehicles in region North. The user may expect that such a difference also holds at more detailed levels. But suppose at a more detailed level, it is found in the datawarehouse instance that for years 2007 and 2008, it is the other way around: Sales of red vehicles are greater than sales of blue ones for region North. It can thus be recommended a query dealing with the *sales of red or blue vehicles in the region North for years 2007 or 2008*.

Third, it could be recommended a query that interested users other than the current user, but whose session is similar to hers. Suppose it is found, for instance in the server log, a session that is very similar to the current session. This logged session investigated sales of red or blue vehicles in various French regions and cities, and it ended with a query asking for *Sales of red vehicles in Marseille in 2006, 2007, 2008, 2009*. This query can then be recommended to the current user.

Finally, a combination of the approaches above can be used, for instance to recommend a query that was posed in the past by a user investigating the same unexpected data as the data the current user is investigating. For instance, the following query can be recommended: *Sales of red or blue vehicles in the regions North or South for years 2006 to 2009*. This query shows a significant deviation from the user's expectation.

We develop this scenario in Section 6 to illustrate the approaches used to generate such recommendations.

3 Recommendations in databases

In this section, we briefly recall the classical viewpoint on recommendations (Adomavicius and Tuzhilin, 2005), and its adaptations to databases.

3.1 Recommender systems

A recommender system is typically modelled as follows. Let I be a set of items and U be a set of users (mostly customers). Let f be an utility function with signature $U \times I \rightarrow R$ for some totally ordered set R. Recommending s' to u is to choose for the user u the item s' that maximizes the user's utility, i.e., $s' = argmax_I f(u, i)$. The function f can be represented as a matrix $M = U \times I$, that records for any user u in U, any item i in I, the utility of i for u, that is $f(u, i)$. The problem of recommending items to users is that this matrix is both very large and very sparse. Thus, many methods have been proposed for estimating the missing ratings.

In general, these methods are categorized (Adomavicius and Tuzhilin, 2005) into: (i) Content-based, that recommend items to the user u similar to previous items highly rated by u, (ii) Collaborative, that consider users similar (i.e. having similar profiles) to the one for which recommendations are to be computed as a basis for estimating its ratings and (iii) Hybrid, that combine content-based and collaborative ones.

Note that Adomavicius et al. (2011) propose a multidimensional generalisation of this basic two-dimensional formulation, especially to support profiling and contextualisation. The view-point we propose in Section 4 can be seen as tailoring this generalisation to datawarehouse peculiarities.

3.2 Recommendation in databases

Recently, to our knowledge, there has been only two attempts to formalize database query recommendations for exploration (Chatzopoulou et al., 2009; Stefanidis et al., 2009). It is important to see that given the context of database exploration, a direct transposition of the users $\times$ items matrix is not relevant. Even assimilating customers with database users and items with database queries raises questions, as illustrated by the fact that databases queries can always be combined, or constructed from user interests, whereas this usually makes no sense with items.

For Chatzopoulou et al. (2009), the problem is viewed as a sessions $\times$ tuples matrix. With this approach, content-based, collaborative and hybrid methods have been developed.

Stefanidis et al. (2009) propose a users $\times$ queries matrix measuring the usefulness of a query q to a user u. This utility is equal to the number of times user u has posed the query q. It is claimed that the technique proposed by Chatzopoulou et al. (2009) falls into this category, although conceptually the setting seems different since Chatzopoulou et al. (2009) consider a sessions $\times$ tuples matrix.

Khoussainova et al. (2010) and Akbarnejad et al. (2010) focus on fragments (attributes, tables, joins and predicates) of queries and consider thus sessions $\times$ query fragments matrix. Starting from a query fragment q, the system searches in the query log similar fragments Q'. Akbarnejad et al. (2010) recommend queries of the log containing Q'. Khoussainova et al. (2010) recommend the most probable fragments of Q' knowing that the initial fragment is Q.

Finally, we remark that for both these works, a fixed database instance is assumed and nothing is proposed to take into account its evolution (it would make sense to consider the database instance even in the history-based approach proposed by Stefanidis et al. (2009)).

3.3 Describing and classifying recommendation approaches

Stefanidis et al. (2009) propose a very interesting taxonomy of database recommendation techniques. There are three categories: (i) 'Current-state' approaches exploiting the content and schema of the current query result and database instance, (ii) 'History-based' approaches using the query logs and (iii) 'External sources' approaches exploiting resources external to the database. Current-state approaches can be based either on (i) the local analysis of the properties of the result of the posed query or (ii) the global analysis of the properties of the database. In both cases systems exploit (i) the content and/or (ii) the schema of the query result or the database. Hybrid methods combining local and global analysis or content and schema can be computed. Stefanidis et al. (2009) focus on the 'Current-state' approach and propose various techniques to recommend queries. Surprisingly, the taxonomy proposed by Stefanidis et al. (2009) does not include a category for hybrid techniques mixing current-state and/or history-based and/or external source techniques.

Golfarelli (2010) quickly reviews methods to relieve users from the burden of tedious analytical query formulation (called query personalisation), and proposes 4 criteria to categorise them:

– Formulation effort: some approaches require the user to manually specify preference criteria for each query, while in others the best personalization criteria are inferred from the context and the user profile.
– Prescriptiveness: some approaches use personalization criteria as hard constraints that are added to a query while in other as soft ones: tuples that satisfy as much preference criteria as possible are returned even if no tuples satisfies all the preferences.
– Proactiveness: distinguishes the approaches that propose new queries based on the navigation log and on the context (but that does not execute them), with respect to those that change the current query or post process its results before returning them to the user.
– Expressiveness: personalization criteria have different expressivities and can be differently combined.

In Section 7, we will use these categories and criteria to characterise the methods we survey.

4 Recommendation for datawarehouses

In this section we formalize the problem of recommending queries for datawarehouse exploration as a function taking into account various parameters including past sessions, datawarehouse instance, etc. We first motivate this formalization by looking at peculiarities of datawarehouse exploration, and then discuss this viewpoint w.r.t. those proposed for database query recommendations (Chatzopoulou et al., 2009; Stefanidis et al., 2009).

4.1 Peculiarities of datawarehouse exploration

As evidenced by e.g., Chaudhuri and Dayal (1997); Sarawagi et al. (1998); Rizzi (2007) basic peculiarities of typical datawarehouse exploration can be summarized by:

1. A datawarehouse is a read-mostly database and its instance has an inflationist evolution (data are added, never or very seldom deleted). Thus it is quite common that a user issues the same sequence of queries more than once, for instance from one year to another. Therefore, if an analysis is conducted year x, it would make sense, if it is found that the same analysis is started year $x + 1$, to recommend the queries launched year x adapted to the data of year $x + 1$.

2. A datawarehouse is a database shared by multiple users whose interest may vary over time. It is argued in Bellatreche et al. (2005); Rizzi (2007); Golfarelli and Rizzi (2009); Rizzi (2010) that user preferences are of particular importance in datawarehouse exploration. It would therefore be important to issue recommendations computed from other users' habits (e.g., in a collaborative filtering fashion) and at the same time respecting the user interests and privacy.

3. A datawarehouse has a particular schema that reflects a known topology, often called the lattice of cuboids, which is systematically used for navigation (Han and Kamber, 2000). Rollup and drilldown operations that allow to see facts at various levels of detail are very popular in this context. In addition, it is often assumed that the description of the levels of detail (the dimensions table) is the part of the datawarehouse instance that can fit in main memory, which is relevant for efficiency purpose in the case of on-line recommendation computation. Recommendations should be computed by taking into account this topology and the semantics attached to the OLAP operations.

4. A typical analysis session over a datawarehouse is a sequence of queries that has a sense w.r.t. some expectations. For instance, the user may assume a uniform distribution of the data (Sarawagi, 1999, 2000) or that two populations follow the same distribution (Ordonez and Chen, 2009). Sessions (as sequences of queries) are of particular importance in this context since by this sequence the user navigates to discover valuable insight w.r.t. her expectations or assumptions. Thus sessions, and more precisely the logical connections between consecutive queries, must be treated as first class citizens when computing recommendations (note that this viewpoint is consistent with the viewpoint adopted for query recommendation in the web (Baraglia et al., 2009)).

4.2 Datawarehouse query recommendation

Let a datawarehouse query (or query for short) be any query expressible in a given language for manipulating multidimensional data (for instance, the MDX query language). Let a session be a sequence of queries and a log be a set of sessions. A datawarehouse instance is classically a set of $n+1$ relation instances where n instances play the role of dimensions and one instance is the fact table. A user profile is any information allowing to define an order over the tuples in the fact table. Let an expectation function be any function on a datawarehouse instance that produces a score (often a real number).

We formalize query recommendations for datawarehouse exploration as a function $Recommend(L, cs, I, P, f)$ that has the following parameters:

- L: A set of sessions (the query log),
- cs: A particular session (the current session),
- I: A datawarehouse instance,
- P: A user profile,
- f: An expectation function.

These parameters allow to cover Current-state (I and f), History-based (L and cs), and External sources (P) approaches (Stefanidis et al., 2009).

The function $Recommend$ returns an ordered set of pairs as recommendations, each pair composed of a query with its associated rating indicating the relevance of the query for the current session. Note that it is out of the scope of this paper to detail the internal structure of $Recommend$.

With this formalization, an explicit user rating for queries is not assumed. It is only considered that once a query is part of a session, it is relevant for the session. Note that queries that do not appear in the former sessions (the log) nor in the current session may be constructed by $Recommend$.

4.3 Discussion

This formalization is well adapted to describe existing proposals (see Sections 5 and 6). It differs from the formalizations of Chatzopoulou et al. (2009); Stefanidis et al. (2009) by the following aspects:

- Sessions are viewed as first-class citizens, and a clear distinction is made between the current session and the past sessions.
- The topology of the database and user expectations are taken into account with the parameter f and the datawarehouse instance I.
- With a sessions $\times$ tuples matrix (Chatzopoulou et al., 2009), only the interest of individual tuples are taken into account for computing recommendation. Thus, groups of tuples can not be rated.
- With a users $\times$ queries matrix (Stefanidis et al., 2009) it cannot be taken into account the fact that a user can launch a sequence of queries, or that user interest for a query may vary.

5 The existing approaches

In this section, we survey the methods that recommend queries for helping the user who poses a sequence of queries over a datawarehouse instance for exploration purposes.

To the best of our knowledge, there are two types of works that investigate query recommendations for datawarehouse exploration: Those that addressed explicitly the issue of recommending datawarehouse queries (Bellatreche et al., 2005; Giacometti et al., 2008, 2009a; Jerbi et al., 2009; Golfarelli and Rizzi, 2009; Golfarelli et al., 2011), and those, less recent, that did not describe themselves as recommendation techniques (Cariou et al., 2008; Sapia, 1999, 2000; Sarawagi, 1999, 2000; Sathe and Sarawagi, 2001). However, the latter suggest a new query based on the user former queries, and thus can also be viewed as recommendation techniques. Note that Bentayeb and Favre (2009), although claiming to recommend queries, suggest changes in the instance (by proposing a new hierarchy), instead of, strictly speaking, recommending a precise query.

These methods are categorized according to the parameters that they use when expressed as calls to the $Recommend$ function presented in the previous section. First, we found methods that exploit information external to the datawarehouse (in our case a user profile). This category resembles the *external sources* category proposed by Stefanidis et al. (2009). Second, the methods that rely on expectations on the data. As such, this category has no correspondence in the taxonomy proposed by Stefanidis et al. (2009). Third, methods leveraging query logs, corresponding to the *history-based* category of Stefanidis et al. (2009). Finally, the fourth category is that of hybrid methods. This leads us to four approaches: Methods exploiting a profile (presented in Section 5.1), methods based on expectations (presented in Section 5.2), methods exploiting query logs (presented in Section 5.3) and hybrid methods (presented in Section 5.4).

5.1 Methods exploiting a profile

The works in this category (Jerbi et al., 2009; Bellatreche et al., 2005) suppose that a profile is provided together with the current session. A profile expresses user preferences over the tuples of the fact table. These methods do not consider any expectations nor log, but take the datawarehouse instance into account (more precisely for Bellatreche et al. (2005) only dimension tables are considered). Thus the call to the $Recommend$ function would be the following: $Recommend(\emptyset, cs, I, P, \emptyset)$. where cs is the current session, I is a datawarehouse instance and P is a profile.

For all these methods, the profile P and the instance I are used to modify the current query (i.e., the last query of cs). The queries that are the result of this modification constitute the recommendation.

Technically speaking, Jerbi et al. (2009) propose that a profile consists of a set of preference predicates that can be added to the current query if they are consistent with it. In that case, the rating output by the $Recommend$ function estimates the interest of a query q for the session cs by evaluating the proportion of relevant preferences that are added to the current query to form q.

Bellatreche et al. (2005) propose to construct a recommended query q by using the elements of the profile P that guarantee that (i) q is included (in the classical sense of query inclusion) in the current query, (ii) q only fetches preferred facts w.r.t. P and (iii) q respects some visualization constraints. We note that the method of Bellatreche et al. (2005) differs with the one of Jerbi et al. (2009) by the fact that the former computes a recommended query that is included in the current query, whereas it is not necessary the case for the latter. Inspired by Koutrika and Ioannidis (2005), the technique proposed by Bellatreche et al. (2005) is called query personalisation (see Bentayeb et al. (2009) for an overview) instead of query recommendation.

The technique of Golfarelli and Rizzi (2009); Golfarelli et al. (2011) also personalises a query, that intends to compute a subquery of the current query. This work is close to Bellatreche et al. (2005) but the approach differs in the sense that it is inspired by that of Kießling (2002) where preferences are used to annotate the query, and uses a richer preference model. In this case, the recommended query is the subquery that returns a non empty preferred answer.

The advantages of the methods in this approach is that recommendations are computed w.r.t. both a profile and the user sessions. Thus different users will obtain different recommendations.

5.2 Methods based on expectations

The works in this category (Cariou et al., 2008; Sarawagi et al., 1998; Sarawagi, 1999, 2000; Sathe and Sarawagi, 2001) rely on discovery driven analysis, where a model on unseen data is used together with the already seen data, i.e., the results of the launched queries. The strongest deviations to the model are recommended.

We briefly recall the concept of discovery driven analysis. To support interactive analysis of multidimensional data, Sarawagi et al. (1998) introduced discovery driven analysis of OLAP cubes. This and subsequent work resulted in the definition of advanced OLAP operators to guide the user towards interesting regions of the cube, lightening the burden of a tedious navigation. These operators are of two kinds. The first kind tries to explain an unexpected significant difference observed in a query result by either looking for more detailed data contributing to the difference (Sarawagi, 1999), or looking for less detailed data that confirm an observed tendency (Sathe and Sarawagi, 2001). The second kind proposes to the user unexpected data in the cube w.r.t. the data she has already observed, by adapting the Maximum Entropy Principle (Sarawagi, 2000).

Recommendation methods based on discovery driven analysis consist in recommending queries that result in data deviating the most from a model (that we call expectation). More formally, this means that the call to the $Recommend$ function is the following: $Recommend(\emptyset, cs, I, \emptyset, f)$ where cs is the current session, I is the datawarehouse instance and f is an expectation function.

Among the works in this approach, the main difference is the model used, i.e., the nature of the function f. Sarawagi (1999); Sathe and Sarawagi (2001); Sarawagi (2000) rely on the assumption of a uniform data distribution. In that case, f checks whether some parts of the cube (either more detailed data or more aggregated data) respect this assumption and outputs a score indicating the importance of the deviation from this assumption. In this case, $Recommend$ outputs the parts of the cube together with their score computed with the f function.

Cariou et al. (2008) assume a statistical independence of the cube's dimensions. In that case, f checks whether dimensions are independent and outputs a score indicating two dimensions' correlation. $Recommend$ outputs the query that, compared to the last query of cs, details one particular dimension, together with the score for this dimension, computed with the f function.

Note that Cariou et al. (2008) and Sarawagi (1999); Sathe and Sarawagi (2001) compute suggestions using only the current query while Sarawagi (2000) considers every former query result.

5.3 Methods exploiting query logs

The works in this category (Giacometti et al., 2008, 2009a; Sapia, 1999, 2000) suppose that a query log is used to look for similarities between the current session and former sessions, to extract one query as the recommendation. Formally, this means that the call to the $Recommend$ function is the following: $Recommend(L, cs, I, \emptyset, \emptyset)$ where L is a query log and sc is the current session.

The methods in this category mainly differ by the way they consider the similarity between sessions and/or queries. Giacometti et al. (2009a) use the classical Levenshtein (for session) and Hausdorff (for queries) distances. Sapia (1999, 2000) group queries by common projections and selections, and use a Markov model to represent sessions. Giacometti et al. (2008) cluster queries using the Hausdorff distance and detects if the current session is a prefix of some existing session. Sapia (1999, 2000) and Giacometti et al. (2008) identify a matching position for the current session in the closest former session and recommend the query after this position. Giacometti et al. (2009a) recommend the last query of the session that is the closest to the current one. The rating that would estimate the interest of a query for the session is computed based on the proximity of the current session with the log queries (Giacometti et al., 2009a) or based on the probability to have the recommended query following the current query (Sapia, 1999, 2000).

5.4 Hybrid methods

The only work in this category is Giacometti et al. (2009b). In this work, the query log is processed to detect discovery driven analysis sessions. Sessions are associated with a goal, and

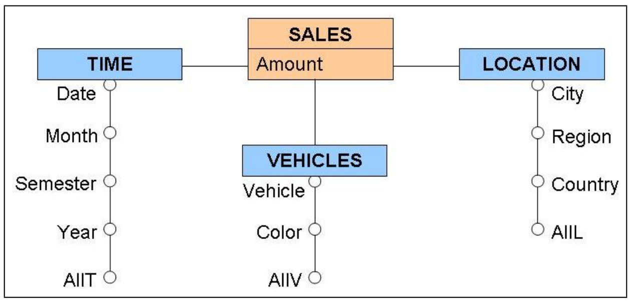


FIG. 2 – *Schema of the datawarehouse.*

recommendations are queries of former sessions having the same goal as that of the current session. This means that the call to $Recommend$ is the following: $Recommend(L, sc, I, \emptyset, f)$, where L is a query log, sc is the current session, I is the datawarehouse instance and f is an expectation function. More precisely, the model of data and function f are as in Sarawagi (1999); Sathe and Sarawagi (2001); Sarawagi (2000). The log is processed to discover pairs of facts that show a significant difference. f is used to detect in the query results of the logged queries those results that strongly deviate from the model. The score is computed accordingly. The main difference with the methods in Sarawagi (1999); Sathe and Sarawagi (2001); Sarawagi (2000) is that only the log is searched for interesting deviations.

6 Case study

In this section, we situate each of the four approaches presented in the previous section in the context of the scenario sketched Section 2. Note that for presentation purposes, the approaches are presented informally on a toy example to give the flavour of the approach without overwhelming the reader with technical details. Most often, a simplified version of the techniques proposed is used in the example.

6.1 The scenario

We consider the following typical case of a datawarehouse exploration by a sequence of queries. The datawarehouse schema is given Figure 2. For the sake of simplicity, we consider a fixed datawarehouse instance I and we focus on only 11 queries noted q_0 to q_{10} in what follows. As in MDX, the de facto standard for querying cubes, each query is given under the form of a set of cell references to extract from the cube, and is presented as a Cartesian product of members per dimension (respectively Vehicles, Time and Location).

$$\begin{array}{lllll}
q_0 = & \{Red,\ Blue\} & \times \{2006,\ 2007,\ 2008,\ 2009\} & \times \{North,\ South\} \\
q_1 = & \{AllV\} & \times \{AllT\} & \times \{France\} \\
q_2 = & \{Red,\ Blue\} & \times \{AllT\} & \times \{North,\ South\} \\
q_3 = & \{Red,\ Blue\} & \times \{2006,\ 2007,\ 2008,\ 2009\} & \times \{Paris\} \\
q_4 = & \{Red,\ Blue\} & \times \{2008\ Sem1,\ 2008\ Sem2\} & \times \{Paris,\ Marseille\} \\
q_5 = & \{AllV\} & \times \{AllT\} & \times \{France, Spain, England\} \\
q_6 = & \{Red,\ Blue\} & \times \{AllT\} & \times \{France\} \\
q_7 = & \{Red,\ Blue\} & \times \{2006\} & \times \{North\} \\
q_8 = & \{Red,\ Blue\} & \times \{AllT\} & \times \{Marseille\} \\
q_9 = & \{Red,\ Blue\} & \times \{AllT\} & \times \{Paris,\ North,\ South\} \\
q_{10} = & \{Red,\ Blue\} & \times \{2009\} & \times \{Paris,\ North,\ South\}
\end{array}$$

Suppose the current user is associated with a profile P indicating that the city $Paris$ should be taken into account if her query deals with region $North$, and that year 2009 is her preferred year. We consider that the queries were used in three previous sessions s_1, s_2, s_3 that constitute the query log L, and in the current session cs as follows:

- $s_1 = \langle q_5, q_6, q_2, q_3, q_0 \rangle$
- $s_2 = \langle q_4, q_6, q_2, q_3 \rangle$
- $s_3 = \langle q_1, q_2, q_4 \rangle$
- $cs = \langle q_1, q_2 \rangle$

Note that queries q_7 to q_{10} do not appear in the previous sessions, they will be constructed by some of the methods illustrated.

We now present what query each of the approaches would recommend in that context for the current session. In what follows, we consider that the scores are in $[0, 1]$.

6.2 Methods exploiting a profile

The call to the recommend function is $Recommend(\emptyset, cs, \emptyset, P, \emptyset)$. In this case, the recommend function constructs queries by combining elements from the profile with the last query of the current session cs. $Recommend$ computes a score for each constructed queries. In our example, this score is based on the number of members from the profile in the query. For q_9 it is $\frac{1}{2}$ because this query deals only with '2009' (i.e., only 1 element out of 2 possible is taken into account). For q_{10} it is 1 since this query deals with 'Paris' and '2009'. Thus the output of $Recommend$ is the set $\{(q_{10}, 1), (q_9, 1/2)\}$.

6.3 Methods based on expectations

Consider the result of q_2 presented in Figure 1 showing a significant difference in sales of red or blue vehicles in the North. A possible expectation would be that this difference is equally distributed at more detailed levels.

The call to the recommend function is $Recommend(\emptyset, cs, I, \emptyset, f)$. In our example, f is a function that, applied on two facts of a query result, computes to what extends pairs of facts of I that detail the two facts of the result deviate from the expectation.

$Recommend$ examines detailed data of I to discover pairs of facts contributing to this difference. It is found that sales of Red or Blue vehicles in 2006 in the North on the one hand and sales of Red or Blue vehicles in Marseille on the other hand deviates from the expectation.

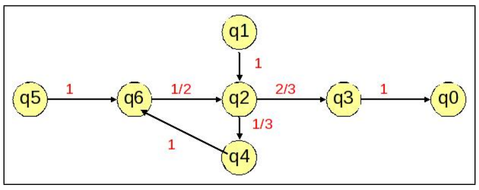


FIG. 3 – *The Markov model of the log*

SALES (Amount)	Red	Blue
2006, North	10 000	50 000
Marseille	10 000	40 000

For the sales of Red or Blue vehicles in 2006 in the North, the difference is 1 to 5 and for the sales of Red or Blue vehicles in Marseille, it is 1 to 4. This difference is only 1 to 2 in the result of query q_2. Hence, two queries are constructed to represent these difference at more detailed levels, namely q_7 and q_8. The score of q_7 is $1 - (\frac{1}{5}/\frac{1}{2}) = 0.6$ and that of q_8 is $1 - (\frac{1}{4}/\frac{1}{2}) = 0.5$ Thus, $Recommend$ outputs $\{(q_7, 0.6), (q_8, 0.5)\}$.

6.4 Methods exploiting query logs

This approach leverages the query log L, i.e., sessions s_1, s_2, s_3, to extract a query as recommendation for session cs. The call to $Recommend$ is $Recommend(L, cs, I, \emptyset, \emptyset)$. We briefly describe two different methods in this category. The first one models the log by a Markov model, and the second one computes distances between sessions.

6.4.1 Using a Markov model

In this method, $Recommend$ constructs the Markov model given Figure 3. Each node is a query from the log and the weighted edge between queries q and q' corresponds to the probability of finding q' after q in the sessions of the log.

In our example, $Recommend$ computes a score for each query of the log that corresponds to the probability of obtaining it after the current query q_2 (the last query of the cs). For the log, this score is $\frac{2}{3}$ for query q_3, $\frac{1}{3}$ for query q_4 and 0 for all other queries. Thus $Recommend$ outputs $\{(q_3, \frac{2}{3}), (q_4, \frac{1}{3})\}$ as result.

6.4.2 Using a distance between sessions

In these techniques, a similarity between sessions is computed to find in the log L the sessions closest to the current session cs. In our example, this similarity is computed w.r.t. a distance that gives the minimal number of operations on the sessions (say, add a query to a session, remove a query from a session or substitute a query by another one) to transform cs into s_1 or s_2. For instance, to transform cs into s_1, it is necessary to add q_5, substitute q_1 by q_6, and add q_3 and q_0, i.e., four operations are necessary. Once the closest sessions to cs is found, a particular query of each session is elected to be the recommendation. In our example, suppose that the recommended query is the last query of the session, and the score for the query is the similarity between cs and the session closest to cs containing the recommended queries. Here, the closest session to cs is s_3, since it takes 1 operation to transform cs into s_3. The similarity is computed as 1 minus (the similarity between sc and s_3 over the maximum difference between session). The recommended query is q_4 with a score of $(1 - \frac{1}{3})$. Thus $Recommend$ outputs $\{(q_4, \frac{2}{3})\}$.

6.5 Hybrid methods

This approach leverages both the log and what is done in the current session, by discovering if the expectation of the current query was the same as some queries in the log. The call to $Recommend$ is $Recommend(L, cs, I, \emptyset, f)$. In this example, f is the same as the one used in the example describing the method based on expectation: Applied on two facts, it looks for pairs of facts that deviate the most from the expectation that the difference in the result of q_2 is equally distributed at more detailed levels. The only difference is that it does not look directly in the datawarehouse instance I but in the result of the queries logged in L.

In the log, only session s_1 is such that (i) the same difference as the one of cs is investigated and (ii) a query result shows a significant deviation at a more detailed level w.r.t. this difference (in our example query q_0). Note that q_0 contains the same pair of fact as q_7 and thus has the same score. Thus $Recommend$ outputs $\{(q_0, 0.6)\}$.

7 Conclusion

In this paper, we survey the problem of recommending a datawarehouse query to a user to guide her during her exploration. We propose to see the problem of recommending a datawarehouse query for exploration purposes as the call to a function $Recommend$ that computes a set of queries and associated ratings given a query log, a session, a user profile, a datawarehouse instance, and an expectation function. The rating computed indicates the usefulness of each query for a session. With this viewpoint, we categorized the existing methods into four approaches: (1) Methods exploiting a profile, (2) methods based on expectations, (3) methods exploiting query logs and (4) hybrid methods. We illustrate them in a simple case study. This categorisation is summarized in the table below:

	(1)	(2)	(3)	(4)
L			✓	✓
cs	✓	✓	✓	✓
I	✓	✓	✓	✓
P	✓			
f		✓		✓

Note that these four categories can also been described, although less precisely, in terms of the categorisation proposed by Stefanidis et al. (2009), as illustrated below:

		(1)	(2)	(3)	(4)
current-State	local				
	global	✓		✓	
	hybrid		✓		✓
history-based	query-based				
	user-based				
	hybrid			✓	✓
external source		✓			

Finally, note that all the approaches surveyed, except that of Golfarelli and Rizzi (2009); Golfarelli et al. (2011), are proactive, prescriptive and require a low formulation effort.

We believe that the field of recommending datawarehouse queries is still in its infancy, as evidenced by the fact that no techniques have yet been proposed that leverage together all the possible parameters of the recommend function. Note also that most of the recommendations given as example in Section 4.1 still cannot be computed with the existing approaches. Many methods can be developed on the basis of the existing approaches identified in this paper. In particular, it will be interesting to see how the techniques proposed by Chatzopoulou et al. (2009); Stefanidis et al. (2009) can be adapted to the datawarehouse context.

An important challenge is to assess the quality of the recommendations computed, which supposes the involvement of real users on real cases, to constitute a baseline. We see this as a major difficulty due to the amount and sensitivity of the information needed, which include real data, user profiles and expectations, as well as past and current analyses.

Another challenge is to propose a strategy for choosing which particular approach to adopt in a given context. Finally, it would also be interesting to investigate the need for an even more general framework than the one proposed in this paper, for instance by including parameters like a set of datawarehouse instances or a set of user profiles.

References

Adomavicius, G. and A. Tuzhilin (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng. 17*(6), 734–749.

Adomavicius, G., A. Tuzhilin, and R. Zheng (2011). Request: A query language for customizing recommendations. *Information Systems Research 22*(1), 99–117.

Akbarnejad, J., M. Eirinaki, S. Koshy, D. On, and N. Polyzotis (2010). SQL QueRIE Recommendations: a query fragment-based approach. In *PersDB*.

Baeza-Yates, R. A. (2010). Query intent prediction and recommendation. In *RecSys*, pp. 5–6.

Baraglia, R., F. Cacheda, V. Carneiro, D. Fernandez, V. Formoso, R. Perego, and F. Silvestri (2009). Search shortcuts: a new approach to the recommendation of queries. In *RecSys*, pp. 77–84.

Bellatreche, L., A. Giacometti, P. Marcel, H. Mouloudi, and D. Laurent (2005). A personalization framework for olap queries. In *DOLAP*, pp. 9–18.

Bentayeb, F., O. Boussaid, C. Favre, F. Ravat, and O. Teste (2009). Personnalisation dans les entreôts de données : bilan et perspectives. In *5èmes Journées Francophones sur les Entrepôts de Données et l'Analyse en ligne (EDA 2009)*, pp. 7–22.

Bentayeb, F. and C. Favre (2009). Rok: Roll-up with the k-means clustering method for recommending olap queries. In *DEXA*, pp. 501–515.

Cariou, V., J. Cubillé, C. Derquenne, S. Goutier, F. Guisnel, and H. Klajnmic (2008). Built-in indicators to discover interesting drill paths in a cube. In *DaWaK*, pp. 33–44.

Chatzopoulou, G., M. Eirinaki, and N. Polyzotis (2009). Query recommendations for interactive database exploration. In *SSDBM*, pp. 3–18.

Chaudhuri, S. and U. Dayal (1997). An Overview of Data Warehousing and OLAP Technology. *SIGMOD Record 26*(1), 65–74.

Giacometti, A., P. Marcel, and E. Negre (2008). A framework for recommending OLAP queries. In *DOLAP*, pp. 73–80.

Giacometti, A., P. Marcel, and E. Negre (2009a). Recommending multidimensional queries. In *DaWaK*.

Giacometti, A., P. Marcel, E. Negre, and A. Soulet (2009b). Query recommendations for olap discovery driven analysis. In *DOLAP*, pp. 81–88.

Golfarelli, M. (2010). Personalization of olap queries. In *6èmes journées francophones sur les Entrepôts de Données et l'Analyse en ligne (EDA 2010)*.

Golfarelli, M. and S. Rizzi (2009). Expressing olap preferences. In *SSDBM*, pp. 83–91.

Golfarelli, M., S. Rizzi, and P. Biondi (2011). myOLAP: An approach to express and evaluate OLAP preferences. *IEEE TKDE, to appear*.

Han, J. and M. Kamber (2000). *Data Mining: Concepts and Techniques*. Morgan Kaufmann.

Jerbi, H., F. Ravat, O. Teste, and G. Zurfluh (2009). Preference-based recommendations for olap analysis. In *DaWaK*.

Khoussainova, N., M. Balazinska, W. Gatterbauer, Y. Kwon, and D. Suciu (2009). A case for a collaborative query management system. In *CIDR*.

Khoussainova, N., Y. Kwon, M. Balazinska, and D. Suciu (2010). Snipsuggest: Context-aware autocompletion for sql. *PVLDB 4*(1), 22–33.

Kießling, W. (2002). Foundations of preferences in database systems. In *VLDB*, pp. 311–322.

Koutrika, G. and Y. E. Ioannidis (2005). Personalized queries under a generalized preference model. In *ICDE*, pp. 841–852.

Ordonez, C. and Z. Chen (2009). Evaluating statistical tests on olap cubes to compare degree of disease. *IEEE Transactions on Information Technology in Biomedicine 13*(5), 756–765.

Rizzi, S. (2007). Olap preferences: a research agenda. In *DOLAP*, pp. 99–100.

Rizzi, S. (2010). New frontiers in business intelligence: Distribution and personalization. In *ADBIS*, pp. 23–30.

Sapia, C. (1999). On modeling and predicting query behavior in OLAP systems. In *DMDW*, pp. 2.1–2.10.

Sapia, C. (2000). Promise: Predicting query behavior to enable predictive caching strategies for OLAP systems. In *DaWaK*, pp. 224–233.

Sarawagi, S. (1999). Explaining differences in multidimensional aggregates. In *VLDB*, pp. 42–53.

Sarawagi, S. (2000). User-adaptive exploration of multidimensional data. In *VLDB*, pp. 307–316.

Sarawagi, S., R. Agrawal, and N. Megiddo (1998). Discovery-driven exploration of OLAP data cubes. In *EDBT*, pp. 168–182.

Sathe, G. and S. Sarawagi (2001). Intelligent rollups in multidimensional OLAP data. In *VLDB*, pp. 531–540.

Stefanidis, K., M. Drosou, and E. Pitoura (2009). "You May Also Like" Results in Relational Databases. In *PersDB*.

Résumé

De nombreuses données sont stockées dans les entrepôts de données. Celles-ci sont naviguées et explorées à des fins d'analyses. Ce n'est que récemment que le problème de recommander une requête à un utilisateur interrogeant un entrepôt de données a attiré l'attention. Dans cet article, nous proposons un cadre formel simple pour exprimer des recommandations de requêtes sur un entrepôt de données. Nous proposons de voir le problème de recommandation de requêtes pour l'exploration comme une fonction calculant un ensemble de requêtes et les estimations associées à partir d'un log de requêtes, d'une session, d'un profil utilisateur, d'une instance de l'entrepôt et d'une fonction de prévision. L'estimation calculée indique l'utilité de chaque requête pour une session. Avec ce point de vue, nous passons en revue et classons les quelques techniques qui, à notre connaissance, ont été proposées et nous les illustrons avec une étude de cas.