Information Access Based on User Preferences

Nicolas Spyratos
Laboratoire de Recherche en Informatique
Bat. 490 Université Paris sud 11
Orsay, France
spyratos@lri.fr

ABSTRACT

In this paper, our basic thesis is that information is useful to the casual user only if it can be retrieved from an information source easily and the retrieved results are presented in a manner easy to exploit or 'digest' by the user. Keeping with this spirit, we present an incremental approach to querying tabular data (such as electronic catalogues), taking into account user preferences. In our approach the answer set of the query is partitioned into blocks, and the blocks are presented to the user one by one: the first block contains the best answers with respect to the user's preferences, the second block contains the second best answers, and so on. The presentation of blocks can stop at any time the user decides to do so.

1. INTRODUCTION

With the advent of the Internet and the Web, tremendous amounts of information are available today to ever increasing numbers of users. This availability of information to casual users has spurred a shift from system centred to user centred or 'personalized' information access.

Indeed, information is useful to the casual user only if it can be retrieved easily and the retrieved results are presented in a manner easy to exploit or 'digest' by the user. Keeping with this spirit, we present an incremental approach to querying tabular data (such as electronic catalogues), taking into account user preferences.

Incorporating user preferences in the dialogue between the user and the information system is one way of achieving personalization. In our approach, during this dialogue, the user submits a query Q together with preferences, online, and the system rewrites Q into a sequence of sub-queries Q1→Q2→ … →Qn whose answers contain the information retrieved in decreasing order of preference. The presentation of sub-query results can stop at any time the user decides to do so.

Our approach addresses the above problem in the context of tabular data. Such data is quite common today, especially in applications concerning electronic commerce. For example Autoreflex is an internet company mediating the selling of used cars through an electronic catalogue containing a reference number for each available car and the main characteristics of the car (model, colour, year, mileage, etc.). The company site is the following:

http://www.google.fr/search?hl=fr&q=Autoreflex& meta=

Information retrieval from such tables is done through Boolean combinations of keywords, therefore rather easy to do by the casual user. However, such tables usually contain thousands or even tens of thousands of entries, so the answer set of a query can be:

- either very small, thus unsatisfactory for the user (the usual solution in this case is to enlarge the query)

- or very large, thus difficult to exploit by the user (the usual solution in this case is to narrow the query)

Unfortunately, the solution to either of these problems can lead to the other problem!

In this paper we address the problem of very large answer sets and we propose a solution based on rewriting of the user query (using the user preferences) as mentioned earlier. More precisely, the questions that we answer are the following:

how can a user express preferences online?

how can preferences be used to rewrite the user query?

how can we embody the rewriting in the query language?

We call *preference based query*, or simply *preference query*, an ordinary query together with a set of preferences expressed by the user, online (i.e. the user inputs both, a query and a set of preferences).

The specification and evaluation of preference queries has received considerable attention in the past several decades, mainly in the area of decision support. However, the use of preferences for ranking query answers in the area of databases is quite recent and their embodiment in the query language presents a number of difficult problems.

Roughly speaking, preferences are distinguished with respect to their nature and with respect to their persistence in time. In terms of their nature preferences are further distinguished in quantitative and qualitative preferences. A *quantitative preference* (or absolute preference) is expressed by a number on a scale (thus capturing intensity of desire). For example, "I like BMWs 80%", or "I like VWs 70%" are quantitative preference expressions (see for example [10]). Quantitative preferences are difficult to express by the casual user but easy to compute by the system (through mining of query logs). A *qualitative preference* (or relative preference) is expressed by comparison (see for example [12]). For example, "I like BMWs more than VWs" is a qualitative preference (note that no intensity of desire is indicated). Qualitative preferences are easy to express by the casual user.

In terms of their persistence or duration in time, preferences are further distinguished to long term preferences and short term preferences. A *long term preference* is either discovered unobstrusively by the system (by mining query logs) or declared explicitly by the user; in both cases the preferences are stored in the so called "user profile". A *short term* preference is expressed explicitly by the user, online, together with the query.

We note that the nature and the duration in time are orthogonal characteristics of preferences. In this work we focus on short term, qualitative preferences

With respect to previous work [14, 20], the present paper contributes in the following ways: (a) it provides a new definition of answer to a preference query, by taking into account the whole answer set (b) it introduces a query rewriting technique inspired from [14].

In what follows, in section 2, we explain the basic concepts through examples; in section 3 we give formal definitions; in section 4, we present the basic design choices of an interface under development and explain how a user interacts with it; finally, in section 5, we offer some concluding remarks and outline possible extensions of this work.

2. PREFERENCE QUERY EXAMPLES

Consider the table T(Serial, Model, Colour, Mileage, Price, Year) of

Figure 1, that we shall use in all our examples. This table describes used cars that are for sale (over the Internet). Each car is described by its serial number, model, colour, mileage, price and year (for simplicity, we denote serial numbers by integers).

When searching for a car, users specify their request as a Boolean combination of elementary conditions of the form $A=v$, $A\neq v$ or $A\leq v$, where A is a column heading (also called an *attribute*) and v is a value of A. As an example, consider the following query:

$Q= [(\text{Model}= \text{BMW}) \vee (\text{Model}= \text{VW})] \wedge (\text{Mileage}\leq 40000 \text{ Km})$

To find the answer to this query we must first find all serial numbers that correspond to BMWs, and all serial numbers that correspond to VWs, then take the union of these two sets, and finally intersect the result found with the set of serial numbers that correspond to cars with a mileage at most 40000 miles:

$$\text{ans}(Q) = (\{1, 2, 6\}\cup \{3, 5, 8, 9\})\cap\{1, 3, 5, 6, 7, 8, 9\}$$
$$= \{1, 3, 5, 6, 8, 9\}$$

As the table might contain a huge number of entries, one problem is that the size of the answer set might be too large to exploit by the average user (i.e. the cars of actual interest to the user might be buried in a large number of uninteresting cars).

One solution to this problem is to present the cars of the answer set in a decreasing order with respect to user preferences. The user can then inspect the most interesting cars first, and stop inspection of the answer set when the cars become less and less interesting. However, to produce such an ordering of the answer set, the system must have access to user preferences, and one way to do this is to have the user declare his preferences *online*, together with the query; one talks then of preference based queries, or *preference queries* for short.

The important thing to stress here is that the preferences are declared by the user in order to influence the presentation of the answer set. For example, consider the following declaration of preference over the attribute Model:

$\text{P.Model: BMW} \rightarrow \text{VW}$

This declaration is taken to mean that, with respect to Model, the user prefers BMWs to VWs. We would like the query Q, seen earlier, together with the above preference, to return a result showing the BMWs *before* the VWs. In other words, we would like the answer to be presented to the user as follows:

$\text{Ans}(Q, \text{P.Model}) = \{1, 6\}\rightarrow\{3, 5, 8, 9\}$

It is important to note that the answer to the query Q, processed alone (i.e. without preferences), and the answer to query Q processed together with the above preference contain the *same* serial numbers. The difference lies in the fact that, in presence of the declared preference, the answer set of Q is partitioned into two blocks (i.e. two subsets) *ordered* so that the first block contains BMWs only (serials 1 and 6), while the second block contains VWs only (serials 3, 5, 8 and 9).

Therefore, in this paper, the answer to a preference query is defined to be a sequence of data blocks, where each block contains data that are more interesting (in terms of the declared preferences) than the data in the following block. In this way, the user can inspect the blocks of the answer set one by one and stop inspection at any point at which he feels satisfied by the data already inspected. We are interested in the computation of such block sequences, and their presentation to the user, when data collections are modelled as tables and preferences as binary relations over attribute values.

In this first, simple example that we have just seen, the preference P.Model involves just one pair of values of the attribute Model. Clearly, the user might want to express more than one such pair, as in the following example:

P.Model: BMW$\rightarrow$VW, BMW$\rightarrow$Honda, VW$\rightarrow$Honda

This time, it is less obvious how to compute the sequence of blocks that answers the preference query.

Moreover, the user might want to express preferences over an attribute of cars not present in the query Q, as in the following example:

P.Colour : Red$\rightarrow$Yellow, Black$\rightarrow$White, White$\rightarrow$Yellow

Another possibility is that the user might want to express preferences by combining values from two different attributes, as in the following example:

P.{Model, Colour}: Red$\wedge$VW$\rightarrow$Yellow$\wedge$BMW

(meaning that the user prefers red BMWs to yellow VWs)

Finally, the user might want to express preferences over two or more columns *independently* of one another, as in the following example:

P.Model: BMW$\rightarrow$ VW, BMW$\rightarrow$Honda

P.Color : Red$\rightarrow$Yellow, Black$\rightarrow$White

In this case, the user might also want to declare *priorities* over the attributes, as in the following example:

Priorities: P.Model$\rightarrow$P.Color

(meaning that Model is more important than Colour, and therefore preferences over Model carry more weight than those over Colour)

We note here that the block sequence that answers a preference query in presence of both, preferences *and* priorities, is more complex to compute, in general. Indeed, in the previous example, one will have to infer precedence over pairs of Model-Color values from the given preferences over Model and over Color.

As we can see from all these different forms that preferences can take, computing the sequence of blocks that answers a preference query can be very involved.

Serial	Model	Colour	Mileage	Price	Year
1	BMW	Black	35000	3800	2002
2	BMW	Black	48000	4500	2001
3	VW	Red	30000	3500	2003
4	Honda	Blue	63000	2900	2000
5	VW	White	26000	3300	2000
6	BMW	Green	16000	5700	2004
7	Toyota	Black	12000	6300	2006
8	VW	Red	34000	5600	2003
9	VW	Yellow	13000	7200	2007

Figure 1. A Used Car Table

3. SPECIFICATION AND EVALUATION OF PREFERENCE QUERIES.

In this section we give the formal definition of a preference query and its answer, as well as an algorithm for the evaluation of the answer. Evaluation is done by rewriting the preference query <Q, Preferences, Priorities> into a sequence of ordinary queries Q_1, Q_2, …, Q_n, such that the answers to the queries of the sequence produce the sequence of blocks answering the preference query.

Let R(Tid, A_1, .., A_n) be a table, in the sense of the relational database model, where each of the attributes Tid, A_1, .., A_n is associated with a set of values, or *domain*. The table of

Figure 1 is an example of such a table. Each line of R (also called a *tuple* of R) consists of values taken from the corresponding domains. We assume that the attribute Tid is a key of the table, thus acting as a *tuple identifier* (or as a "surrogate" for the whole tuple); for example, in

Figure 1, Serial is the key. Given a tuple t of R, we use the notation t.A_i to denote the value of tuple t on attribute A_i; for example, in

Figure 1, if t is the third line of the table then t.Colour = Red and t.Mileage= 30000.

Definition 1. A *preference* over attribute A is defined to be just a pair of values (v, v') from the domain of A. Such a pair is denoted as v$\rightarrow$v' and interpreted as "v is *preferred* to v' ", or "v *precedes* v' ".

The set of all preferences expressed by a user over an attribute A is called a *preference relation* over A. A preference relation P over A is denoted as P.A. For example, in

Figure 1, the following is a preference relation over the attribute Color:

P.Colour: Red$\rightarrow$Yellow, Red$\rightarrow$White, Black$\rightarrow$White

Definition 2. Given two tuples t, t' of R, and a preference relation P.A, we say that "t *is preferred to* t" with respect to P.A, or that "t *precedes* t" with respect to P.A, if t.A$\rightarrow$t'.A is in P.A.

For example, with respect to the preference relation P.Colour given above, we have the following precedence over tuples in

Figure 1: 3$\rightarrow$5, 3$\rightarrow$9, 1$\rightarrow$5. Note that no other precedence between tuples is possible to infer from the given preference relation.

Definition 3. A *preference query* over R is a pair <Q, P.A> such that:

1) Q is an *ordinary query* over R, that is a Boolean combination of elementary conditions of the form A=v, A≠v or A≤v, where A is an attribute and v is a value of A

2) P.A is a preference relation over attribute A

The *answer* to a preference query, denoted ans(<Q, P.A>), is defined to be a sequence R_0, R_1, .., R_m, R_{m+1} of sets of tuples, such that:

1/ the sets R_0, R_1, .., R_{m+1} form a partition of ans(Q) (i.e. they are mutually disjoint and their union is ans(Q))

2/ R_0 contains the "best" tuples, that is for every tuple t in R_0 there is no tuple s in R such that s$\rightarrow$t; and for each i= 1, 2, ..m, and for each tuple t in R_i there is a tuple s in R_{i-1} such that s$\rightarrow$t with respect to P.A

3/ R_{m+1}= ans(Q) \ $R_0 \cup \ldots \cup R_m$

Note that the block R_{m+1} in the above definition contains all tuples in the answer of Q that cannot be compared to other tuples.

In what follows, for notational convenience, we shall drop the attribute names whenever they are easy to understand from context. For example, we shall write

Q= BMW$\vee$VW instead of

Q= (Model = BMW) $\vee$ (Model = VW)

Following this convention, let us illustrate Definition 3 using a very simple example. Consider the preference query <Q, P.Colour>, defined over the table of

Figure 1 as follows:

Q= (BMW$\vee$VW)

P.Colour: Red$\rightarrow$Yellow, Red$\rightarrow$White, Black$\rightarrow$White

First, let us observe that the answer to Q, denoted ans(Q), contains all cars that are of interest to the user; it is precisely this set that we need to partition into a sequence of subsets R_0, R_1, .., R_m, R_{m+1} that will constitute the answer to <Q, P.Colour> (see point 1/ in the above definition). To compute R_0, we look at the preference relation P.Colour and we observe that Red and Black are the only two colours that are not preceded by any other colour (and none of them precedes the other). Therefore, all tuples of ans(Q) that refer to either a red or a black car are the "best" with respect to the preferences in P.Colour (see point 2/ of the above definition). More formally, this is expressed as follows:

R_0= ans(Q)$\cap$[ans(Red)$\cup$ans(Black)]

Now, if we call Q_0 the ordinary query whose answer is R_0, then Q_0 can be expressed in terms of Q, Red and Black as follows:

Q_0= Q$\wedge$(Red$\vee$Black)

A similar reasoning shows that the ordinary query whose answer is R_1 is defined as follows:

Q_1= Q$\wedge$(Yellow$\vee$White)

There are no more colours to consider in the preference relation P.Colour, therefore these two steps correspond to point 2/ in the above definition. As for point 3/, we have:

R_2= ans(Q) \ ($R_0 \cup R_1$)

(note that R_2 contains all tuples that are not possible to compare with other tuples, with respect to P.Colour).

Now, if we call Q_2 the ordinary query whose answer is R_2, then we can express Q_2 in terms of Q, Q_0 and Q_1 as follows:

Q_2= Q $\wedge$ $\neg$($Q_0 \wedge Q_1$)

As a result, the sequence of ordinary queries Q_0, Q_1, Q_2 is such that the answers to its queries produce the sequence of blocks answering the preference query of our example. Clearly, if the preference relation P.Colour is more complex, then we need an algorithm in order to produce the sequence of queries that answers the preference query. In the remaining of this section we present such an algorithm.

First, let G be an acyclic binary graph, and define the *rank* of a node t as follows:

if t is a root of G then rank(t)= 0

else rank(t)= the length of a maximal path among all paths from a root of G to t

Next, let us denote by B_i the set of nodes with rank i, and let m be the maximal path length among all paths starting from a root. Then it is rather easy to see that the sequence $B_0, B_1, .., B_m$ has the following properties:

1/ $B_0, B_1,.., B_m$ form a partition of the set of nodes of G

2/ for each i= 1, 2,.., m, and each node in B_i there is an antecedent s of t in B_{i-1} (i.e. there is a node s in B_{i-1} and an arc $s \rightarrow t$ in G)

3/ for each i= 0, 1, 2,.., m, there is no arc of G connecting two nodes of B_i

To find the sets $B_0, B_1, .., B_m$ one can use the following algorithm (which is a variant of the well known topological sorting algorithm):

```
Algorithm Ordered-partition(G)

Input: An acyclic graph G
Output: A sequence B_0, B_1, .., B_m of sets of nodes
Method:
Aux:= G; i:= 0 ;
while Aux≠∅ do
  begin  B_i:= {r / r is a root of Aux};
           output B_i;
           delete from Aux all roots and
         all arcs emanating from the roots;
            i:= i+1
  end;
```

The complexity of this algorithm is linear in n+a, where n is the number of nodes and a is the number of arcs of G.

Now, in a preference query <Q, P.A>, the preference relation P.A can be represented as a binary graph that we shall denote by G(P.A). With this observation at hand, the following algorithm produces the sequence of queries $Q_0, Q_1, .., Q_m, Q_{m+1}$ that answers the preference query <Q, P.A>:

```
Algorithm Evaluate-Pref-query

Input: A preference query <Q, P.A> such that the
graph G(P.A) is acyclic.
Output: The sequence Q_0, Q_1, .., Q_{m+1} answering the
preference query
Method:
1/   Ordered-partition(G(P.A))
          {the output is a sequence B_0, B_1, ..,
                   B_m of sets of values of A}
2/    For each i=0, 1, .., m do
        begin
          Q'_i := conjunction of all values of A
                                        in B_i;
          Q_i := Q∧Q'_i;
          output Q_i
        end
3/   Q_{m+1} := Q∧ ¬ (Q_0∨Q_1∨…∨Q_m);
   output Q_{m+1}
```

So far, we have considered that the preference relation P.A is expressed over a single attribute. Clearly, if the preference relation is expressed over two or more attributes the way of defining the answer to the preference query remains the same. In other words, if instead of having P.A we now have P.{A, B} then the only difference is that each node of the graph G(P.{A, B}) is a conjunction of two values, a value of A and a value of B (instead of being just a single value). For example, consider a preference query <Q, P.{Model, Colour} with the following preferences:

P.{Model, Colour}:
$Red\wedge VW \rightarrow Yellow\wedge BMW$, $Black\wedge VW \rightarrow Yellow\wedge BMW$

Then the answer to the preference query is the following sequence:

$Q_0 = Q\wedge[(Red\wedge VW)\vee(Black\wedge VW)]$

$Q_1 = Q\wedge(Yellow\wedge BMW)$

$Q_2 = Q\wedge \neg (Q_0\vee Q_1)$

However, things become more complex when two or more preference relations are declared together with the ordinary query Q, that is when the preference query has the form <Q, {P.A_1, .., P.A_k}>. Clearly, in this case we can proceed as in the case above, provided that we can derive a preference relation P.{A_1, .., A_k} from the given preference relations P.A_1, .., P.A_k. However, in order to do this, we need to know whether the preference relations P.A_1, .., P.A_k all carry the same weight or there are priorities among them; a *priority* is a linear ordering over {P.A_1, .., P.A_k} and it is declared by the user together with the preference relations P.A_1, .., P.A_k. The following definitions state how the preference relation P.{A_1, .., A_k} is derived from the given preference relations P.A_1, .., P.A_k. In these definitions, "Pa" stands for "Pareto" and "Pr" stands for "Prioritized".

Pareto preference relation:

For all tuples s and t in R, $s \rightarrow_{Pa} t$ if and only if $s.(A_1\ldots A_k) \neq t.(A_1...A_k)$ and either $s.A_i = t.A_i$ or $s.A_i \rightarrow_i t.A_i$, i=1,.., k

We treat the relation Pa up to tuple equivalence, where equivalence is defined as follows: $s \equiv_{Pa} t$ if and only if $s.(A_1..A_k) = t.(A_1..A_k)$.

Prioritized preference relation:

Let the preference relations $P.A_1, .., P.A_k$ be prioritized as follows: $P.A_1 \rightarrow P.A_2 \rightarrow \ldots \rightarrow P.A_k$, where we use the arrow $\rightarrow$ to also show priority. Under this assumption, we have the following definition:

For all tuples s and t in R, $s \rightarrow_{Pr} t$ if and only if $s.(A_1 \ldots A_k) \neq t.(A_1 \ldots A_k)$ and either $s.A_1 \rightarrow_i t.A_1$ or $[(s.A_1 = t.A_1$ and $s.(A_2..A_k) \rightarrow_{Pr} t.(A_2..A_k)]$

We treat the relation Pr up to tuple equivalence, where equivalence is defined as follows: $s \equiv_{Pr} t$ if and only if $s.(A_1..A_k) = t.(A_1..A_k)$.

We note that the well known lexicographic ordering is a special case of prioritized relation Pr as defined above. Indeed, the lexicographic ordering is a prioritized ordering with the additional assumption that the domain of each of the attributes $A_1, ..., A_k$ is totally ordered (i.e. given any two values v and v' of attribute A_i, either $v \rightarrow v'$ or $v' \rightarrow v$)

3. THE INTERFACE

The interface that we have implemented allows users to input queries, preferences and priorities in a user friendly manner; and to control the presentation of the answer set, through two buttons, “Next” and “Stop”, in an interactive way, as follows:

- enter query, preferences, and possibly priorities;
- activate the button “Next” (if you wish to see the next block) else activate the button “Stop”.

More specifically, our interface allows users to perform the following tasks:

Searching the Table: The user can enter an ordinary query Q; this is done by forming elementary conditions of the form A=u or A=v or A=w, and connecting them using Boolean connectives chosen from a menu.

Entering Preferences**:** The user can enter preferences and possibly priorities, together with the query Q. To declare a preference relation, the user first selects an attribute from a menu containing all attributes, and then declares pairs of values of that attribute; and if more than one preference relation is declared with the same query Q then the user is asked to select a priority (either Pareto or Prioritized) from a popup menu.

Controlling the presentation: In the presence of preferences, the user can control the presentation of the answer set by activating two buttons, “Next” and “Stop”. The button “Next” is pressed when the user wishes to see the next block of tuples in the sequence of blocks answering the preference query; the button “Stop” is pressed when the user wishes to terminate inspection of the answer set.

In each task, the user has the possibility to undo an action.

The third task above (controlling the presentation) is a basic feature of the interface. Indeed, it is expected that, as the blocks are presented in decreasing order of preference, the user will find the data of interest quite quickly (i.e. in the first few blocks). As a result, the interface will not have to execute all the sub-queries $Q_0, Q_1, \ldots, Q_m, Q_{m+1}$, in the answer to the preference query. Although we have not yet conducted real life experiments, we expect this feature to contribute significantly in enhancing user satisfaction, as well as improving the performance of the interface (in terms of savings in computation time).

In designing the interface, a major decision to be made was whether the preference relation should satisfy certain properties. For example, in the related literature, the preference relation is always assumed to be transitive. In designing our interface we made no assumption whatsoever as to the nature of the preference relation. In other words, we let users declare just any preference relation they wish (i.e. just any set of pairs of values from a selected attribute).

However, as we have seen in the previous section, the algorithm that computes the answer to a preference query *does* assume the preference relation to be acyclic (in fact, this is an indispensable assumption in order for the algorithm to work). On the other hand, as we would like the user to be free to declare any desirable preference, the set of preferences declared by the user might contain cycles. So the question arises as to how the system could cope with the presence of cycles in the preference relation.

Intuitively, a cycle means that all nodes in the cycle are equally preferred. If the system detects cycles in the preference relation submitted by the user, then these cycles can be handled in one of two ways, as follows:

Dialogue with the user: The cycles are presented to the user, and the user is asked to break them (possibly by modifying the declared preferences).

Automatic Processing: The system processes the cycles without help from the user, by considering all nodes on a cycle as being equivalent, thus “coalescing” all nodes of a cycle into a single node. We note that, formally, cycle equivalence in a preference relation P is defined as follows: (a) $x \equiv x$, for all attribute values appearing in P and (b) $x \equiv y$, if x and y are on the same cycle. Then instead of P one works with the quotient relation $P/\equiv$.

To apply either of these two ways of processing preference relations containing cycles we need an efficient algorithm for finding all cycles in a graph;

and although there are such algorithms in the literature, we have designed and implemented a novel algorithm that outperforms existing algorithms [15-19].

4. CONCLUDING REMARKS

We have presented an approach for processing preference queries over tabular data. In designing the interface, particular care was taken to make the task of the user as simple as possible. In this respect, the following two features of the interface are important: (a) the user can declare as preferences just any set of pairs of values over *any* attribute of the table, and (b) the user can control the presentation of the answer set.

The implementation of the interface is now completed but several aspects need to be improved. One major feature currently missing is the possibility to create intervals over attributes with ordered domains. For example, it would be interesting to be able to create intervals over attributes such as Mileage or Price. Such intervals, if given appropriate names (e.g. low, medium, high) would be much more convenient for the user to use, instead of using specific numbers.

As a final remark we would like to emphasize that there is an important difference between the preference queries that we study here and the Order-by instruction of SQL. Indeed, using the Order-by instruction, one can ask the system to return the results of a query in an ascending or descending order, following the *predefined* order of some attribute domain (e.g. the predefined order over the domain of attribute Price is that of the integers). However, in the preference queries considered in this paper, it is the user that *inputs* an order for the attribute domain - an order expressing the user's preferences; and in fact, the order that the user inputs might contradict the predefined order of the attribute domain. Additionally, some attribute domains have no predefined order; for example, the domains of attribute Model or Colour are unordered so the Order-by instruction simply doesn't apply to such attributes. In contrast, preference queries as explained here apply on *any* attribute, independently of whether its domain is ordered or not.

1. REFERENCES

[1] Agrawal, R., and Wimmers, E. L. A Framework for Expressing and Combining Preferences. ACM SIGMOD (Dallas, USA), 2000, 297-306.

[2] Balke, W.-T., Güntzer U., and Kießling W. On Real-time Top k Querying for Mobile Services. International Conference on Cooperative Information Systems, Irvine, USA, 2002.

[3] Börzsönyi, S., Kossman, D., and Stocker, K. The Skyline Operator. In Proceedings of the 17th International Conference on Data Engineering (ICDE), Heidelberg, 2001.

[4] Boutilier, C., Brafman, R., Hoos, H., and Poole, D. Reasoning with conditional ceteris paribus preference statements. In UAI-99, pages 71-80, 1999.

[5] Bruno, N., Gravano, L., and Marian, A. Evaluating Top-k Queries over Web-Accessible Databases. ICDE, 2002, 369-279.

[6] Chomicki J, Iterative Modification and Incremental Evaluation of Preference Queries. FoIKS 2006: 63-82

[7] Chomicki, J. Querying with Intrinsic Preferences. In Proceedings of the 8th International Conference on EDBT, Prague, Czech Rep., 2002, 34-51.

[8] Chomicki, J. Semantic optimization of preference queries. In 1st Int. Sym. on Appl. of Constraint Databases, Springer (LNCS 3074), 2004.

[9] Fagin R et al., Comparing Partial Rankings, SIAM J. Discrete Mathematics

[10] Fagin R. et al. Comparing and aggregating rankings with ties. In PODS, 2004.

[11] Fagin R., Kumar R., and Sivakumar D. Comparing top k lists. In SODA, 2003.

[12] Kießling, W., and Köstler, G. Preference SQL Design, Implementation, Experiences. In Proceedings of 28th International Conference on Very Large Data Bases, Hong Kong, China, 2002, 990-1001.

[13] Manzini Paola & Mariotti Marco, 2003. "How vague can one be? Rational preferences without completeness or transitivity," Game Theory and Information 0312006, EconWPA, revised 16 Jul 2004.

[14] N. Spyratos, V. Christophides, P. Georgiadis, M. Nguer, Semantics and Pragmatics of Preference Queries in Digital Libraries, Knowledge Media Science Workshop, October 2-5, 2006, Meiningen Castle, Germany (Also: LRI Research Report N°1478, November 2007

[15] An optimal algorithm for cycle breaking in directed graphs T. Orenstein, Z. Kohavi I. Pomeranz . Journal of Electronic Testing: Theory and Applications, 7, 71-81 (1995)

[16] D.B. Johnson. Find all the elementary circuits of a directed graph. J. SIAM, 4:77-84, 1975

[17] J.C. Tiernan. An efficient search algorithm to find the elementary circuits of a graph. Comm. ACM, 13:722-726, 1970.

[18] A new way to enumerate cycles in graph Hongbo Liu; Jiaxin Wang - Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services (AICT-ICIW 2006). 19-25 Feb. 2006 Page(s): 57 – 57

[19] P. Mateti and N. Deo. On algorithms for enumerating all circuits of a graph. SIAM J.

Comput., 5:90-99,1976

[20] P.Georgiadis, I.Kapantaidakis, M.Nguer, N.Spyratos, V.Christophides, Efficient Rewriting Algorithms for Preference Queries, 24th International Conference on Data Engineering (ICDE’08), April 7-12, 2008, Cancun, Mexico