CPref-SQL: A Query Language Supporting Conditional Preferences

Sandra de Amo
Federal University of Uberlandia
Uberlandia, Brazil
deamo@ufu.br

Marcos Roberto Ribeiro
Federal University of Uberlandia
Uberlandia, Brazil
mribeiro@pos.facom.ufu.br

ABSTRACT
Nowadays, the need for incorporating preference querying in database technology is a very important issue in a variety of applications ranging from e-commerce to personalized search engines. A lot of recent research work has been dedicated to this topic in the artificial intelligence and database communities. Several formalisms allowing preference reasoning and specification have been proposed in the AI field. On the other hand, in the database field the interest has been focused mainly in extending standard SQL with preference facilities in order to provide personalized query answering. In this paper, we propose to build a bridge between these two approaches, by using a logic formalism originally designed to specify and reason with preference in order to extend SQL with conditional preference constructors. Such constructors allow to express a large class of preference statements with a ceteris-paribus semantics.

Categories and Subject Descriptors
H.2.3 [Database Management]: Languages—Query Languages; I.2.4 [Artificial Intelligence]: Knowledge Representation Formalisms and Methods

General Terms
Algorithms, Languages, Theory

Keywords
Query Language, Preferences, SQL Extension, Relational Algebra

1. INTRODUCTION
Recently, a lot of interest arose in the artificial intelligence and database communities concerning the topic of preference elicitation, modelling and reasoning. In fact, due to the huge amount of information users are faced up to daily, the development of formalisms for preference specification and reasoning as well as tools allowing to cope with real user preferences turn out to be essential tasks in AI and DB fields. A lot of work has been done in this area so far by researchers in both communities. From a AI point of view, the interest is centered mainly in developing powerful mechanisms for preference reasoning [3, 4, 18]. On the other hand, database researchers are concerned specially in developing preference query languages with high declarative and expressive power for personalized database applications [12, 6, 10, 13, 15, 14]. In this context, a lot of work has been done so far concerning the so-called skyline queries [6, 5, 2] and more generally, the pareto preference queries [12, 16, 17]. In this paper, we propose CPref-SQL, an extension of SQL able to express conditional preference queries (cp-queries for short). These queries incorporate the usual hard constraints (declared inside the WHERE clause) as well as soft constraints specified by a set of preference rules (cp-rules for short). A typical example of such a query is as follows.

EXAMPLE 1. Let us suppose we are given a database relation MOVIES storing information about movies with attributes T (for Title), D (for Director), G (for Genre), Y (for Year). The following statements express my preferences about movies: (1) For Woody Allen’s films (wa), I prefer comedies (c) better than thrillers; (2) For Robert Altman’s films (ra), I prefer thrillers (t) to comedies; (3) For comedies I prefer the more recent ones, those produced after 1990. I would like to list the titles of the films which most fulfill my wishes among those stored in the database, produced after 1975. The corresponding CPref-SQL query is:

CREATE PREFERENCE MyPrefs AS
  IF D = wa THEN (G = c) > (G = t) AND
  IF D = ra THEN (G = t) > (G = c) AND
  IF G = c THEN (Y \geq 1990) > (Y < 1990)

SELECT T FROM MOVIES
WHERE Y > 1975
ACCORDING TO PREFERENCES MyPrefs
Here, the hard constraint is Y > 1975 and the soft constraints are given by the rules MyPrefs.
Cp-queries differ from pareto queries ([11, 12]) in three main aspects: (1) local preferences over domain values are not absolute, that is, they depend on how other attributes are instantiated; (2) The semantics of cp-queries is based on the ceteris paribus semantics: tuples differing on attributes not appearing in a rule statement are incomparable by this rule. This is not the case with pareto queries; (3) A conditional preference order corresponding to a set of rules is obtained by taking the transitive closure of the union of the preference orders specified by each rule individually. The result may be inconsistent, that is, it can be inferred that “I prefer X better than X”! That is not the case with the special of the pareto queries of [11, 12]: The result of pareto accumulation of weak orders over the attribute domains is a strict partial order over the set of tuples [6]. So, consistency issues have to be taken into account when dealing with cp-queries. We achieve this by following the ideas introduced in [18], in the context of preference modelling and reasoning.

The main contribution of this paper consists in the following three points: (1) The introduction of cp-queries which incorporate conditional preferences as soft constraints; (2) An extension of the classical relational algebra of SQL with a new operator SelectBest(Perm, R), where Perm is a set of cp-rules and R is a database relation; (3) An algorithm BNL∗ for implementing the SelectBest operator. The semantics of SelectBest follows the lines of the BMO query model (Best Matches Only) used in [11, 6, 7]: First find all perfect matches satisfying the hard and soft constraints. If this set is empty, then all tuples satisfying the hard constraints are returned.

This paper is organized as follows. In Section 2, we discuss some related work. In Section 3, we briefly present the logic formalism introduced [18] for specifying and reasoning with conditional preferences which constitutes a necessary background for our work. In Section 4, we present the language CPref-SQL, an extension of standard SQL allowing to express user preference statements. We also present in this section the syntax and semantics of the core CPref-SQL algebra which extends the relational algebra with a SelectBest operator. In Section 5, we discuss the implementation details of the SelectBest operator and CPref-SQL query evaluation. Finally, in Section 6 we conclude the paper by discussing some future work.

2. RELATED WORK

The research literature on preference reasoning and eliciting over objects is extensive. The approach of CP-Nets [3] uses a very simple graphical model which captures users qualitative conditional preference over objects, under a ceteris paribus semantics. The approach of TCP-Nets [4] generalizes the CP-Nets by introducing the ability of expressing absolute and relative importance of object attributes. In this paper, we use the approach introduced in [18] for specifying soft constraints inside a CPref-SQL query. This approach uses a logical framework for expressing conditional preference statements. It generalizes the CP-Nets and TCP-Nets. In the database area, the problem of enhancing well-known query languages with preference features has been tackled in several recent and important work in the area. A pioneer work concerns the skyline operator introduced in [2]. In [6], a simple logical framework is proposed for expressing preferences by preference formulae. These formulae are incorporated into relational algebra and into SQL, through the operator winnow parameterized by a preference formula. [12] introduced Preference SQL which extends SQL with some built-in base preference constructors and a pareto constructor. The optimizer uses an efficient rewriting procedure which transforms preference queries into standard SQL queries.

The work presented in [9] is close to ours in the sense that it also proposes a bridge between the treatment of preferences in the AI and DB communities, by transforming the TCP-Net queries of [4] into preference database queries. Although TCP-Nets translate into conditional preference queries with a ceteris paribus semantics, they are less general than the cp-queries we introduce in this paper. Besides, no algorithm for selecting the best tuples has been proposed neither in [9] nor in further work (to the best of our knowledge).

The issue of preference query evaluation has been extensively studied in the literature, in the context of pareto and skyline queries. In [2], the BNL (block-nested loop) algorithm has been introduced for evaluating skyline queries. In [17], the algorithm BNL++ for pareto query evaluation has been proposed. It identifies some important pruning conditions in the better-than graph, which results in a far better performance when compared to the BNL algorithm. Further improvements of the BNL algorithm, in the context of pareto queries evaluation, were achieved in [5] (BBS+, SDC and SDC+). The algorithm BNL∗ we propose in this paper follows the lines of the BNL algorithm and uses the ideas of [1] for transitive closure evaluation.

3. PRELIMINAIRES

We briefly present the main concepts and results of [18] concerning a method for reasoning with preferences. Let $R = \{A_1, A_2, ..., A_n\}$ a relation schema. For each attribute $A_i \in R$, let $dom(A_i)$ be the finite set of values of $X$ (the domain of $X$). The set $\text{Tup}(R) = dom(A_1) \times dom(A_2) \times \cdots \times dom(A_n)$ is the set of all possible tuples over $R$. Tuples are denoted by symbols in boldface $t, a, b, ..., c$. Let $L$ be a set of sets of the form $\varphi : u \rightarrow (A = a) > (A = a') \in \Gamma$. Let $t = (b, c, a, w)$ and $t' = (b, c', a', w') \in \text{Tup}(R)$, where $b$ is a tuple over $\text{Attr}(a)$, $c$ is a tuple over $\text{R} - (A)$ and $a, a' \in dom(A_i)$ for all $i \in \{1, ..., k\}$. Let $\text{W} \subseteq \text{R} - \{A_1, ..., A_n\}$. Such statements are called conditional preference rules or cp-rules for short. The formula $u$ is called the condition of the cp-rule $\varphi$. The set of attributes appearing in $u$ is denoted by $\text{Attr}(u)$. A conditional preference theory (cp-theory) over $R$ is a finite set of statements of $L$.

A conditional preference theory $\Gamma$ over $R$ induces a preference ordering over $\text{Tup}(R)$ as follows:

**Definition 1** (Preference order). Let $\varphi : u \rightarrow (A = a) > (A = a') \in \Gamma$. Let $t = (b, c, a, w)$ and $t' = (b, c', a', w') \in \text{Tup}(R)$, where $b$ is a tuple over $\text{Attr}(a)$, $c$ is a tuple over $R - (\text{Attr}(u) \cup W) \cup \{A\}$, $w$ and $w'$ are tuples over $W$. We say that $t$ is preferred to $t'$ according to $\varphi$. The set of pairs of tuples $(t, t')$ where $t$ is preferred to $t'$ according to $\varphi$ is denoted by $>_\varphi$. We denote by $>_\Gamma$ the transitive closure of the binary relation $\cup_{\varphi \in \Gamma} >_\varphi$.

**Example 2.** Let MOVIES = $\{T, G, D, Y\}$ as in Example 1. Let us assume that $dom(T) = \{T_1, T_2, T_3\}$, $dom(G) =$...
\{c (comedy), t (thriller)\}, \text{sdom}(D) = \{wa (Woody Allen), ra (Robert Altman)\}, \text{dom}(Y) = \{recent', old'\}. The cp-theory \(\Gamma = \{\varphi_1, \varphi_2, \varphi_3\}\) expresses the user preferences of Example 1:

\[ \varphi_1: (D = wa) \rightarrow (G = c) > (G = t)\{(T)\}, \]
\[ \varphi_2: (D = ra) \rightarrow (G = t) > (G = c)\{(T)\}, \]
\[ \varphi_3: (G = c) \rightarrow (Y = old) > (Y = recent)\{(T)\}. \]

We have: \((T_1, t, ra, old)\) is preferred to \((T_2, c, ra, old)\) according to \(\varphi_2\). And \((T_2, c, ra, old)\) is preferred to \((T_3, c, ra, recent)\) according to \(\varphi_3\). Then, using transitivity, we conclude that \((T_1, t, ra, old)\) is preferred to \((T_3, c, ra, recent)\), that is, \((T_1, t, ra, old)\) \(\Gamma\) \((T_3, c, ra, recent)\).

Notice that, for each cp-rule \(\varphi\), the induced preference order \(\succ_\varphi\) has a \textit{ceteris paribus} (“everything else equal”) semantics with respect to the attributes in \(R - (\{A\} \cup W)\). That is, tuples having different values for attributes in \(R - (W \cup \{A\})\) do not coincide in incomparable according to \(\succ_\varphi\). In Example 2, for instance, we can compare tuples \((T_1, t, wa, old)\) and \((T_2, c, wa, old)\) according to \(\succ_\varphi\) because their values for attributes \(D\) and \(Y\) are respectively identical (= \(wa\) and \(old\) respectively). Notice that, according to cp-rules \(\varphi_1, \varphi_2\) and \(\varphi_3\), we may compare tuples for which the values for attribute \(T\) are different.

Unlike pareto preference order, the preference order induced by a cp-theory is not necessarily a strict partial order. In fact, \(\succ_\Gamma\) may contain a pair \((t, t')\), that is, \(\succ_\Gamma\) may not satisfy irreflexibility. Obviously, this situation is not desirable when dealing with preferences in a database context. It is reasonable to require that one may not prefer something better than itself. We say cp-theory \(\Gamma\) is \textit{consistent} if and only if the induced order \(\succ_\Gamma\) is irreflexive. In [18], a sufficient condition for ensuring consistency of a cp-theory is given. This condition involves testing the acyclicity of the \textit{dependency graph} associated to the cp-theory and its \textit{local consistency}.

For lack of space, we do not discuss the consistent test here. For more details, see [18]. In what follows, we will suppose our cp-theories are consistent, that is, the induced order \(\succ_\Gamma\) is irreflexive.

4. CPref-SQL

In this section we introduce an algebraic operator that selects from a given relation the set of the most preferred tuples satisfying a given cp-theory \(\Gamma\). We denote this operator by \textbf{SelectBest}_\(\Gamma\)(\(R\)). The semantics of \textbf{SelectBest}_\(\Gamma\) follows the usual “Best Matches Only” (BMO) query model ([6, 11]): for every instance \(r\) of \(R\).

\[ \textbf{SelectBest}_\(\Gamma\)(\(r\)) = \{t \in r \mid \neg \exists t' \in r, t' \succ_\Gamma t\} \]

Thus, the execution of \textbf{SelectBest}_\(\Gamma\) on an instance \(r\) consists in finding the set of tuples verifying the \textit{soft constraints} \(\Gamma\) (an \textit{optimal} world perspective). If this set is non-empty only these tuples are returned. Otherwise, all non-dominated tuples are returned (a \textit{real} world perspective).

The CPref-SQL is obtained by extending the standard SQL with this new preference constructor. The simple query block of CPref-SQL is given below. Notice that hard and soft conditions are incorporated into a unique simple query block.

\begin{verbatim}
SELECT <attribute-list>
FROM <tables>
WHERE <where-conditions (hard conditions)>
ACCORDING TO PREFERENCES
<lister-of-cprules (soft conditions)>
GROUP BY <attribute-list>
ORDER BY <attribute-list>
\end{verbatim}

A simple execution plan associated to a CPref-SQL block (without aggregate constructors) is shown in Figure 1. \(\sigma\), \(\pi\) and \(\times\) denote the usual relational algebra operators \textit{Selection}, \textit{Projection}, and \textit{Join} respectively.

\begin{figure}
\centering
\begin{tikzpicture}
\node {\pi} ;
\node {SelectBest} [below of = \pi] ;
\node {\sigma} [below of = SelectBest] ;
\node {\times} [below of = \sigma] ;
\node {\ldots} [below of = \times] ;
\end{tikzpicture}
\caption{Execution plan for a CPref-SQL block}
\end{figure}

5. IMPLEMENTATION ISSUES

In [6], in a general framework for expressing preferences, it was shown that a constructor for selecting the best tuples following a BMO query model semantics does not increase the expressive power of the classical relational algebra. That is, such constructor can be translated into a relational algebra expression. In fact, it can even be used to simulate set difference operator. Thus, if we are not concerned with optimization issues, CPref-SQL could be implemented as an intermediate layer between the application and the SQL database system. However, efficient evaluation techniques are desirable in order to execute complex preference queries. We propose the algorithm BNL* described in Figure 2. The general structure of BNL* follows the lines of the \textit{Blocked Nested Loop} (BNL) algorithm [17] and it uses the \textit{Better-than graph} (BTG) to compare tuples \(2\).

The core procedure of BNL* is the \textit{dominance test} which verifies \(t \succ_\Gamma t'\). That is true if there exists tuples \(t_1, t_2, \ldots, t_{n-1}\) and cp-rules \(\varphi_1, \ldots, \varphi_n\) in \(\Gamma\) such that \(t \succ_{\varphi_1} t_1 \succ_{\varphi_2} t_2 \succ_{\ldots, \varphi_n} t'\). So, the dominance test reduces to transitive closure membership testing. Notice that the dominance test for the pareto queries of [11] is far simpler, since it does not involve transitive closure testing.

Before executing algorithm BNL*, the \textit{Better-than graph} (BTG) \(G\) associated to the cp-theory \(\Gamma\) is constructed. In

\footnote{The vertex of BTG are the tuples in the database and the edges stand for the preference relation between the tuples: an edge from tuple \(t\) to tuple \(t'\) means that \(t\) is preferred to \(t'\).}
Input: a relation \( r \), an interval-set labelled BTG \( G \)
Output: a set \( S \) containing the best tuples of \( r \) according to \( G \)

1. \( S := \emptyset \)
2. For all \( t \in r \)
3. If there exists \( t' \in S \) such that \( t' \) dominates \( t \)
4. ignore \( t \)
5. Elseif there exists \( t' \in S \) such that \( t \) dominates \( t' \)
6. eliminate the dominated tuples
7. insert \( t \) into \( S \)
8. Elseif \( t \) is incomparable with all tuples in \( S \)
9. insert \( t \) into \( S \)
10. EndIf
11. EndFor
12. Return \( S \)

**Figure 2: Algorithm BNL**

what follows, we use the well-known notion of postorder traversal of a tree, illustrated in Figure 5.

![Image](https://via.placeholder.com/150)

**Figure 3: The postorder traversal of a tree.**

We use a compression scheme which consists in associating a set of intervals \([n,m]\) to each tuple in the input relation \( r \). The dominance test between two tuples \( t,t' \) will be achieved by comparing the set of intervals associated to these tuples in a simple way. This construction was introduced in [1] as an efficient method to calculate transitive relationships.

1. Consider the Better-Than Graph (BTG) \( G = (V,E) \) of the relation \( \succ_\Gamma \). This graph is obtained as follows: (1) \( V = \text{Tup}(R) \), (2) \( E = \bigcup_{\forall \Gamma \subseteq \Gamma} \succ_\Gamma \) (see Definition 1).

   Notice that, as we only consider acyclic cp-theories, we know that \( \succ_\Gamma \) is a strict partial order, and so, its BTG is acyclic. We can assume that \( G \) consists of only one connected component; disjoint components can be linked together by a virtual root node.

2. Consider the optimal spanning tree associated to \( G \) (we may use algorithm Optimum Tree-Cover of [1]).

3. For each node \( t \) in the spanning tree consider the interval \([i_t,j_t]\), where \( j_t \) is the postorder number of the node \( t \) in the spanning tree and \( i_t \) is the lowest postorder number among its descendents. The postorder numbers of the nodes in a tree are obtained by performing a postorder traversal in the tree. These intervals are called tree intervals.

4. For all nodes \( t \) of \( G \) considered in the reverse topological order do:
   - For each edge \((t,t')\) add the non-tree intervals associated with \( t' \) to the intervals associated with \( t \).
   - After adding an interval to the interval set \( I \) associated to a node, if there exists two intervals \([t_1,t_2]\) and \([t_3,t_4]\) in \( I \) such that \([t_1,t_2]\) is contained in \([t_3,t_4]\) then remove \([t_1,t_2]\) from \( I \).

   In Figure 3, the non-tree intervals (those which have been added and possibly merged) are enclosed with squares.

   The dominance test \( t \succ_\Gamma \ t' \) is achieved as follows: Let \( I(t) \) and \( I(t') \) be the set of intervals of \( t \) and \( t' \) respectively. Then \( t \succ_\Gamma \ t' \) iff there exists \([n_1,n_2]\) \( \in I(t) \) and \([m_1,m_2]\) \( \in I(t') \) such that \( n_1 \leq m_2 < n_2 \).

   The following example illustrates the execution of algorithm BNL.

**Example 3.** Let us consider the CPref-SQL query and the relation schema MOVIES of Example 1 (notice that the cp-theory corresponding to MyPrefs is the set \( \Gamma \) given in Example 2) and the instance \( r \) over MOVIES shown in Table 1.

First of all, the BTG associated to the set of preferences specified in the query is created. This graph is illustrated in Figure 3. Each node in the graph represents a tuple in the set Tup(MOVIES). The arcs of the optimal spanning tree are indicated by solid arrows and the non-tree arcs by dashed arrows. The tuples belonging to \( r \) are enclosed within rectangles.

![Image](https://via.placeholder.com/150)

**Figure 4: The Better-than Graph associated to a cp-theory \( \Gamma \)**

Notice that \( t_6 = (T6,wa,t,\text{recent}) \) is dominated by \( t_8 = (T8,ra,t,\text{recent}) \), since the postorder 2 of \([2,2] \in I(t_6) \) belongs to the non-tree interval \([1,2] \in I(t_8) \). So, \( t_8 \succ_\Gamma \ t_6 \).

Now, let us execute BNL over \( r \).

1. \( S := \emptyset \).
2. \( t_2 \) is inserted into \( S \).
sequences of objects queries involving preferences on temporal extension of SQL (TPrefSQL) which allows to specify an update of the cp-theory. We are also working on a temporal techniques to update the BTG after a database update or theory Γ. An important line of research is to study incremental studying techniques to prune the BTG associated to a cp-

3. \( t_3 \) dominates \( t_2 \) since its postorder number \( 1 \in [1, 6] \in I(t_3) \). So, \( S = \{ t_3 \} \).

4. \( t_6 \) is dominated by \( t_3 \) since its postorder number \( 2 \in [1, 6] \in I(t_3) \). So, \( S = \{ t_3 \} \).

5. Return \( S = \{ t_3 \} \).

BNL⁺ can be adapted in order to select de \( k \) most preferred tuples in a relation \( r \). Thus, CPref-SQL can be extended with a SelectBest \((k, R)\) constructor. The SelectBest operator is a special case where \( k = 1 \). We notice that, unlike the preference constructors of [12], our new operators can appear in a query block nested inside a WHERE clause.

6. CONCLUSION AND FURTHER WORK

In this paper, we have introduced the conditional preference queries (cp-queries), which allow users to specify preferences over the values of attributes depending on the values of other attributes in the tuples. Our specification of cp-queries in the database context is adapted from the logical formalism introduced in [18]. We proposed an extension of SQL with a constructor SelectBest allowing to select the most preferred tuples according to a set of conditional preference rules. We also proposed an algorithm BNL⁺ for implementing the SelectBest operator. This algorithm can be adapted to select the \( k \) best tuples satisfying a set of conditional preference rules. A lot of work remain to be done in order to We are presently working on the implementation of CPref-SQL and studying techniques to prune the BTG associated to a cp-theory \( \Gamma \). An important line of research is to study incremental techniques to update the BTG after a database update or an update of the cp-theory. We are also working on a temporal extension of SQL (TPrefSQL) which allows to specify queries involving preferences on sequences of objects. The logical formalism underlying this temporal query language has been presented in our previous paper [8].

7. REFERENCES


Table 1: Instance \( r \) over MOVIES

<table>
<thead>
<tr>
<th>ID</th>
<th>Title</th>
<th>Director</th>
<th>Genre</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Crimes and Misdemeanors</td>
<td>Woody Allen</td>
<td>Thriller</td>
<td>1989</td>
</tr>
<tr>
<td>2</td>
<td>Mash</td>
<td>Robert Altman</td>
<td>Comedy</td>
<td>1970</td>
</tr>
<tr>
<td>3</td>
<td>Manhattan Murder Mystery</td>
<td>Woody Allen</td>
<td>Thriller</td>
<td>1993</td>
</tr>
</tbody>
</table>


