Handling of NULL Values in Preference Database Queries

Endres Markus and **Roocks Patrick** and **Wenzel Florian** and **Huhn Alfons** and **Kießling Werner** [1]

Abstract. In the last decade there has been much interest in preference query processing for various applications like personalized information or decision making systems. Preference queries aim to find only those objects that are most preferred by the user. However, the underlying data set may contain NULL values which represent *unknown* or *incomplete* data. Most of the existing algorithms for preference query evaluation do not know how to treat these NULL values and consider them worse than any other value. Other algorithms do not allow NULLs in their input data set. However, NULL values are common in data sets and must be considered in preference query evaluation. In this paper we introduce an approach to handle NULL values in preference queries which extends preference algebra, a formal model for preference specification. Our approach can be adopted by all preference query algorithms which rely on strict partial orders, because it does not violate the transitivity relation as other methods do.

1 Introduction

Preferences in databases – as shown by a recent survey [18] – as well as preferences in artificial intelligence and social choice theory (cp. [17]) are a well established framework to create personalized information systems. By using well designed preference models, users can be provided with just the information they need, thereby overcoming the dreaded empty result set and flooding effect [10].

However, the data set behind these information systems may contain *unknown* data, known as NULL values in database systems. NULL is a special marker to indicate that a data value does not exist in the database and therefore represents *missing and inapplicable information*. In standard SQL the handling of NULL values has been the focus of controversy for more than 30 years resulting in a three-valued logic [6]. Hence, comparisons with NULL can never result in either *true* or *false*, but always in a third logical result, *unknown*.

However, the discussion of NULLs in preference database queries is an open issue. Almost all algorithms for preference evaluation (e.g., [1, 5, 7, 14, 16]) rely mainly on two assumptions: First, all preference algorithms assume transitivity in the dominance relation, and second, data is complete, i.e., all dimensions are available for all data objects. The first assumption of dominance transitivity is one of the most used properties in preference algorithms. If a data tuple t_1 dominates tuple t_2 while t_2 dominates t_3, then t_1 dominates t_3, too. Using transitivity, preference query processing algorithms exploit various ways of data pruning and indexing. Obviously, the second assumption of completeness is not practical in a real world database, where NULL values frequently occur, cp. [13].

[1] University of Augsburg, Germany, email: {endres, roocks, wenzel, huhn, kiessling}@informatik.uni-augsburg.de

Table 1. Sample table of hiking tours.

tours	**id**	**length**	**difficulty**	**rating**	**vista**
	1	23.5	medium	4	excellent
	2	NULL	easy	5	bad
	3	NULL	NULL	2	bad
	4	13.1	hard	2	good
	5	7.3	NULL	1	excellent

For example, in a hiking tour database (cp. Table 1) it is highly unlikely that all data for all attributes of a tour are always known. The column *length* contains two NULL entries, because it was not possible to determine the length of the tours. Furthermore users may fill a global database with their own hiking tours. If a hiking tourist wants to set the length but doesn't know it or does not want to rate the difficulty of the tour, he omits the input value. Thus the missing data has to interpreted correctly by setting this value to NULL instead of default value, e.g. 0.

If there are NULLs in the database, how should one compare the unknown to the known values in preference queries? For example, if a users' preference is to find hiking tours with a length of 20 km, how are the given values $\{23.5, 13.1, 7.3\}$ compared to NULL?

One may state that NULL should be always worse than all other values. This would be good for a hiking tourist which is a cautious and accurate person and plans all tours in detail. However, for an user who is adventurous, ready to tackle new challenges and who likes to get surprised by new tours, NULL values in the result of the preference query would be a welcome variety. Hence, this user prefers tours with unknown values over fully documented tours.

The same question also arises for Pareto preference (Skyline) queries [1, 10], where two or more preferences are equally important. In a Pareto preference a tuple t_1 is said to dominate a tuple t_2 if t_1 is better than or equal to t_2 in all dimensions and is strictly better than t_2 in at least one dimension. Unfortunately, with the existence of some incomplete dimensions, we cannot simply use the traditional definition of the dominance relation as it is not immediately clear how to compare an incomplete dimension with a corresponding complete dimension. For example, consider the wish for a tour having a difficulty of 'hard' and a rating of 2. We cannot judge which tuple of $t_1 = (\text{hard}, 2)$ and $t_2 = (\text{NULL}, 2)$ is superior in the first dimension.

The aim of this paper is to extend preference queries to cope with the existence of incomplete data. We provide an approach to handle NULL values in preference queries such that the transitivity relation will be preserved and the assumption of data completeness is not necessary for preference evaluation algorithms. Furthermore we suggest a syntax extension for Preference SQL queries [12] to specify the treatment of NULLs in our preference database system.

The rest of this paper is organized as follows: Section 2 highlights related work. An overview of the used preference model is given in Section 3. Section 4 introduces our NULL value handling in preference algebra. Afterwards, we extend the syntax of the Preference SQL query language in Section 5. Finally, we conclude in Section 6.

2 Related Work

Preference queries are more general than the well known Skyline queries introduced more than ten years ago by Börzsönyi et al. [1]. Skyline queries are a special kind of Pareto preference queries and aim to find tuples which are not dominated by others. Since then, several algorithms have been proposed for preference and Skyline query evaluation that include index-based solutions, pre-sorting and no pre-processing, cp. [1, 5, 7, 16] to name a few. Unfortunately, all these algorithms consider only the case of complete data, i.e. data where all values are known. However, NULL values occur frequently in real life data sets.

Several papers have studied the evaluation of Skyline queries over uncertain (probabilistic) data [15]. Uncertain data in those works is generally caused by data randomness, incompleteness, limitations of measuring equipment, etc. Due to the importance of those applications and the rapidly increasing amount of collected and accumulated data containing uncertainty, analyzing large collections of uncertain data has become an important task. However, how to conduct advanced analysis on uncertain data remains an open problem at large [15].

One of the first works on incomplete data and NULL values was done by Chan et al. [2]. They consider a tuple to dominate another tuple only if a subset of a given size of the dimensions dominates the corresponding dimensions in another tuple. Under this definition, the dominance relation becomes non-transitive. Therefore, traditional preference algorithms cannot be applied.

The closest work to ours is the Skyline querying in the presence of incomplete data [9], which is based on the former mentioned paper. In this work for any two *incomplete* tuples only the common dimensions that are known in both tuples are considered. Among these common dimensions only, they apply the traditional dominance relation to decide which tuple dominates the other, if any. However, this fails if there are no common dimensions. Furthermore, Chomicki rightly asks "What is the right logic for defining such preference relations?", cp. [4].

We introduce an approach of NULL value handling which maintains the transitivity of the dominance relation. Therefore every preference algorithm requiring transitivity can be applied to evaluate preferences on incomplete data.

3 Preferences in Database Systems

Preference modeling has been in focus for some time, leading to diverse approaches, e.g. [3, 10, 11]. We follow the preference model from [11] which is a direct mapping to relational algebra and declarative query languages, e.g., Preference SQL which is discussed in Section 5. It is semantically rich, easy to handle and very flexible to represent user preferences which are ubiquitous in our life.

Formally, a *preference* P on a set of attributes A is defined as $P := (A, <_P)$, where $<_P$ is a strict partial order on the domain of $\mathrm{dom}(A) \times \mathrm{dom}(A)$. For $x, y \in \mathrm{dom}(A)$ the term $x <_P y$ is interpreted as "I like y more than x". We say x and y are *indifferent*, if $\neg(x <_P y) \wedge \neg(y <_P x)$, i.e., neither x is better than y nor y is better than x. Note that the preference order $<_P$ is irreflexive and transitive.

The *Best-Matches-Only*-set (BMO-set) of a preference contains all tuples from a data set that are not dominated w.r.t. the preference. Best-Matches-Only offers a cooperative query answering behavior by automatic matchmaking: The BMO query result adapts to the quality of the data in the database, defeating the empty result effect and reducing the flooding effect by filtering out worse results.

To specify a preference, a variety of intuitive base preference constructors together with some complex preference constructors has been defined. Subsequently, we present some selected preference constructors used in this paper. More preference constructors as well as their formal definition can be found in [10, 11, 12].

3.1 Base Preference Constructors

Preferences on single attributes are called *base preferences*. There are base preference constructors for *discrete* (*categorical*) and for *continuous* (*numerical*) domains. Figure 1 shows the taxonomy of several frequently occurring base preferences [12].

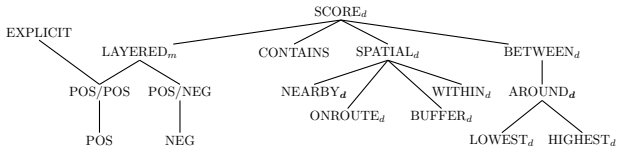

Figure 1. Taxonomy of base preference constructors

Subsequently we describe some numerical base preferences.

Definition 1 (SCORE$_d$ Preference)**.** *Given a scoring function* $f : \mathrm{dom}(A) \to \mathbb{R}_0^+$*, and some* $d \in \mathbb{R}_0^+$*. Then* P *is called a* SCORE$_d$ *preference, iff for* $x, y \in \mathrm{dom}(A)$*:*

$$x <_P y \iff f_d(x) > f_d(y)$$

where $f_d : \mathrm{dom}(A) \to \mathbb{R}_0^+$ *is defined as:*

$$f_d(v) := \begin{cases} f(v) & \text{if } d = 0 \\ \left\lceil \frac{f(v)}{d} \right\rceil & \text{if } d > 0 \end{cases}$$

Note that in the case of $d = 0$ the function $f(v)$ models the *distance* to the best value. That means $f_d(v)$ describes how far the domain value v is away from the optimal value. A d-parameter $d > 0$ represents a discretization of $f(v)$, which is used to group ranges of scores together. The d-parameter maps different function values to a single number. Choosing $d > 0$ effects that attribute values with identical $f_d(v)$ value become indifferent.

The BETWEEN$_d$ preference is a sub-constructor of SCORE$_d$. It expresses the wish for a value between a *lower* and an *upper* bound. A deviation of d does not matter. For BETWEEN$_d(A, [low, up])$ we have $f(v) = \max\{low - v, 0, v - up\}$. Specifying $low = up$ (=: z) in BETWEEN$_d$ we get the AROUND$_d(A, z)$ preference, where the desired value should be z, i.e. $f(v) = |z - v|$. Furthermore, the LOWEST$_d(A)$ and HIGHEST$_d(A)$ constructors prefer the minimum and maximum of the domain of A.

***Example* 1.** The $P_1 := \mathrm{AROUND}_2(\mathrm{rating}, 4)$ preference on Table 1 expresses the wish for a tour rating around 4 where a difference of 2 does not matter. Obviously, the tuple with ID 1 is the most preferred value.

All categorical preferences are sub-constructors of LAYERED$_m$.

Definition 2 (LAYERED$_m$ Preference)**.** *Let $L = (L_1, \cdots, L_m)$ be an ordered list of m sets forming a partition of $\mathrm{dom}(A)$ for an attribute A. The preference P is a* LAYERED$_m(A, (L_1, \ldots, L_m))$ *preference if it is a* SCORE$_d$ *preference with the following scoring function: $f(v) := i - 1 \iff x \in L_i$. For convenience, one of the L_i may be named "*OTHERS*", representing the set $\mathrm{dom}(A)$ without the elements of the other subsets. This implies* OTHERS *contains also* NULL*, if* NULL *is not contained in any other layer.*

Furthermore, sub-constructors of LAYERED$_m$ for frequently occuring cases exist, e.g. POS(A, *POS-set*), which is equal to LAYERED$_2$(A, *POS-set*, OTHERS). It expresses that a user has a set of preferred values, the *POS-set*, in the domain of A. There is also a NEG-preference NEG(A, *NEG-set*). Moreover, it is possible to combine these preferences to POS/POS or POS/NEG. For the POS/POS(A, *POS1-set*, *POS2-set*) preference a desired value *should be* amongst a finite set *POS1-set*. Otherwise it *should be* from a disjoint finite set of *alternatives POS2-set*. If this is also not feasible, better than getting nothing any other value is acceptable. There are many more base preference constructors (cp. Figure 1), all described in [10, 11, 12, 19].

***Example* 2.** Let $P_2 := \mathrm{POS}(\textit{vista}, \{\mathrm{excellent}, \mathrm{good}\})$. That means that we are looking for tours having an excellent or good vista. From Table 1 we get the BMO-set with IDs {1,4,5}.

3.2 Complex Preference Constructors

If one wants to combine several preferences into more *complex preferences*, one has to decide the relative importance of these given preferences. Intuitively, people speak of "this preference is more important to me than that one" or "these preferences are all equally important to me". Equal importance is modeled by the so-called *Pareto preference*.

Definition 3 (Pareto Preference)**.** *In a Pareto preference $P := P_1 \otimes P_2 = (A_1 \times A_2, <_P)$ all preferences $P_i = (A_i, <_{P_i})$ on the attributes A_i are of equal importance, i.e., for two tuples $x = (x_1, x_2)$, $y = (y_1, y_2) \in \mathrm{dom}(A_1) \times \mathrm{dom}(A_2)$ we define:*

$$(x_1, x_2) <_P (y_1, y_2) \textit{ iff}$$
$$(x_1 <_{P_1} y_1 \wedge (x_2 <_{P_2} y_2 \vee x_2 =_P y_2)) \vee$$
$$(x_2 <_{P_2} y_2 \wedge (x_1 <_{P_1} y_1 \vee x_1 =_P y_1))$$

The *Prioritization preference* allows the modeling of combinations of preferences that have different importance.

Definition 4 (Prioritization)**.** *Assume preferences $P_1 = (A_1, <_{P_1})$ and $P_2 = (A_2, <_{P_2})$, then prioritization denoted by $P := P_1 \,\&\, P_2$ is defined as:*

$$(x_1, x_2) <_P (y_1, y_2) \textit{ iff } x_1 <_P y_1 \vee (x_1 =_P y_1 \wedge x_2 <_{P_2} y_2)$$

***Example* 3.** Reconsider the preferences P_1 and P_2 from Example 1 and 2. In the Pareto preference $P := P_1 \otimes P_2$ both preferences are equally important. Tuple 1 dominates tuple 2 and 3, because it is better in both dimensions. Tuple 1 is better than tuple 5 concerning the rating and equal in the vista. Therefore tuple 5 is dominated by tuple 1. Tuple 4 and tuple 2 are indifferent. Tuple 4 is better concerning the rating, but incomparable concerning the vista (*excellent* is not equal to *good*). Therefore, the BMO-set is given by the IDs $\{1, 4\}$.

3.3 Preferences with SV-Semantics

There are situations where indifferent objects should be treated as *substitutable*. That means for base preferences that *all* objects v with equal $f_d(v)$ function value can be designated as *equally good*. This behavior is called *regular Substitutable-Values-Semantics* (SV-semantics). Using regular SV-semantics, all objects with the same $f_d(v)$ value are positioned on the same *level*. Obviously, level 0 contains the perfect matches, higher levels are worse. Having *trivial* SV-semantics only *equal* values are considered as equally good. Following [11] we write $P = C(A, <_P, \cong_P)$ for a preference having any SV relation. We use $\sim_P$ for regular and $=_P$ for trivial SV-semantics.

For base preferences regular SV-semantics does not affect $<_P$ itself, but expresses that it is admissible to substitute values for each other. A complex constructor using $\sim_P$ instead of $=_P$ in its definition (cp. Def. 3 and 4) does affect $<_P$, as we can see in the next example.

***Example* 4.** Consider the Pareto preference $P := P_1 \otimes P_2$ from Example 3. From this example we know that the result of P using trivial SV-semantics is given by the IDs $\{1, 4\}$. Using regular SV-semantics for vista the values *excellent* and *good* are equally good. Since tuple 1 is better than tuple 4 concerning the rating and *excellent* is *substitutable* to *good*, tuple 1 is preferred over tuple 4.

4 NULL Values in Preference Database Queries

In this section we formally introduce the handling of NULL values in preferences. In our proposed approach, NULL is fully integrated in the preference order, i.e. comparisons of NULL and any other value of the domain are possible. To this end we define the NULL-*extended* domain by

$$\mathrm{dom_N}(A) := \mathrm{dom}(A) \cup \{\mathrm{NULL}\}$$

Note that in standard SQL NULLs are not special domain values. A three-valued logic is used, where comparisons with NULL return the third truth value *unknown*. We will use a two-valued boolean logic. An expression $x <_P$ NULL or NULL $<_P x$ with $x \in \mathrm{dom_N}(A)$ is either *true* or *false*. Additionally we require the NULL-extended preference relation $<_p$ to be transitive. Due to these requirements we can use traditional algorithms for the evaluation of preferences.

In the following sections we adapt the preference constructors from Section 3 to the NULL-extended domain. SV-semantics (Section 3.3) are also extended to NULL-values, i.e. the user may specify for which values of x the expression $x \cong_P$ NULL is *true*.

4.1 Insertion Strategies

One possibility to extend preferences to $\mathrm{dom_N}(A)$ is to treat the NULL-value like a value of the original domain, i.e. NULL is *inserted* into the order at the same place as a value from $\mathrm{dom}(A)$. In the case of base preferences, we distinguish between categorical and numerical preferences: For a categorical preference, NULL can be written in the POS-set, OTHERS-set, one of the LAYERED-sets, etc. while for numerical preferences one can either define a NULL-equivalent value (NULL equals 4.5) or place NULL at the top or bottom of the preference order.

Another approach to handle NULL-values is to make the NULL incomparable to all other values, i.e. the expression $x <_P$ NULL is *false* for all x. This models the *missing information* character of the NULL-values: If one knows nothing about a given value, one does not assume any *better than* relations to other values.

Incomparable NULL values are not dominated by any value of $\mathrm{dom}(A)$ and do not dominate any of these values. Hence tuples with NULL-values in the respective attribute always occur in the BMO-set of the preference.

4.2 Extended Categorical Preference Constructors

In the categorical preference constructors, NULL can be used like a usual domain value as shown in the following example:

***Example* 5.** Consider the LAYERED$_m$-preference on attribute A. NULL can be contained in one of the L_i, e.g.

$$\text{LAYERED}_4(\textit{difficulty}, (\{\text{'easy'}\}, \{\text{'medium'}\}, \{\text{'hard'}, \text{NULL}\}, \text{OTHERS}))$$

which means that NULL in the difficulty attribute of the hiking tour is equally disliked as *hard*.

Analogously POS, NEG, POS/POS, etc. are extended in the same manner, i.e. NULL may be written in the POS-set, NEG-setc, etc.

To specify that NULL is incomparable or NULL is placed in the worst layer we introduce a NULL-*handling parameter* for the constructors. Thereby $\text{N}_?$ means NULL *is incomparable to all other values* whereas $\text{N}_{\max}$ places NULL in the worst layer, formally:

Definition 5. *Let C be a preference constructor, A an attribute, X an parameter (Layered-sets, POS-set, etc.) for C and $\cong$ the SV-relation. Then for a preference $P = C(A, X, \cong_P)$ we define:*

1. *Let $P' = C(A, X, \cong_P, \text{N}_?)$, then $<_{P'}$ is given by:*

$$x <_{P'} y = \begin{cases} \text{false} & \text{if } x = \text{NULL} \ \vee \ y = \text{NULL} \\ x <_P y & \text{otherwise} \end{cases}$$

2. *Let $P' = C(A, X, \cong_P, \text{N}_{\max})$. We set the* SCORE*-function (Def. 1) for* NULL *to the maximum of the other values of the domain:*

$$f(\text{NULL}) = \max\{f(v) \mid v \in \mathrm{dom}(A)\}$$

4.3 Extended Numerical Preferences Constructors

For the numerical preference constructors we introduce a constructor which assigns a level or a distance to the NULL-value; additionally NULL may be incomparable, as defined before.

Definition 6. *For a numerical preference constructor C, attribute A, SV-Relation $\cong$ and an optional d-Parameter d and parameter X we define the preference $P = C_d(A, X, \cong_P, N)$, where N may be:*

- $N = \text{N}_?$*: cp. Def. 5, i.e.* NULL *is incomparable to all other values*
- $N = \text{N}_v^{\text{dist}}$*:* NULL *is on distance v.*
- $N = \text{N}_v^{\text{level}}$*:* NULL *is on level v – only if d-Parameter is set with $d > 0$ and regular SV-semantics are used.*

where $v = \max$ means that the $f(\text{NULL})$ is set to the highest level or distance which occurs in $\mathrm{dom}(A)$, cp. Def. 5.

We have the special cases:

- $N = \text{N}_0^{\text{dist}}$*:* NULL *is as good as the best values.*
- $N = \text{N}_\infty^{\text{dist}}$*:* NULL *is the worse than all values of* $\mathrm{dom}(A)$

Thereby *distance* refers to the f-function in Def. 1 whereas *level* refers to the f_d function. In this case we have the equivalences $\text{N}_0^{\text{dist}} \equiv \text{N}_0^{\text{level}}$, $\text{N}_\infty^{\text{dist}} \equiv \text{N}_\infty^{\text{level}}$ and $\text{N}_{\max}^{\text{dist}} \equiv \text{N}_{\max}^{\text{level}}$, where $\equiv$ means that the corresponding preference order is the same.

***Example* 6.** Let $P_3 = \text{AROUND}_{10}(\text{length}, 20, \sim_P, N)$ and consider the tours attribute in the sample data in Table 1. There is no perfect match, i.e. no tour with length 20. Thus for $\text{N}_1^{\text{level}}$ only NULL is in the BMO-set. The length values 23.5 and 13.1 are on level 1 and they are the best matches in $\mathrm{dom}(\textit{length})$, hence for $\text{N}_1^{\text{level}}$ and $\text{N}_?^{\text{level}}$ they are together with NULL in the BMO-set. As the maximum level for P_3 is 2, for $\text{N}_{\max}^{\text{level}}$ and $\text{N}_v^{\text{level}}$ with $v \geq 2$ the NULL value is less preferred than 23.5 and 13.1. In summary we have:

N	BMO-set of values for "length"
$\text{N}_0^{\text{level}}$	$\{\text{NULL}\}$
$\text{N}_1^{\text{level}}, \text{N}_?^{\text{level}}$	$\{\text{NULL}, 23.5, 13.1\}$
$\text{N}_{\max}^{\text{level}}, \text{N}_2^{\text{level}}, \text{N}_3^{\text{level}}, ..., \text{N}_\infty^{\text{level}}$	$\{23.5, 13.1\}$

4.4 Complex Preferences and SV-Semantics

We defined how NULL values are placed in the preference order. Now we consider SV-semantics and complex preferences.

NULL is now a part of the domain and the NULL-extended preferences are still strict partial orders. Therefore the composition of complex preferences can be straight-forward applied to preferences with domain $\mathrm{dom}_{\text{N}}(A)$. For the SV-semantics the same holds: For trivial SV-Semantics $x =_P x$ holds while $x =_P y$ is *false* for $x \neq y$. Note that this implies that NULL $=_P$ NULL is always *true* (in contrast to the trivalent logic in standard SQL). For regular SV-semantics NULL becomes substitutable with all values v having the same level (for $\text{N}_v^{\text{level}}$) or the same distance (for N_v^{dist}). If $N = \text{N}_?$ is used, NULL is not substitutable with any value.

The grouping preference P grouping A evaluates the preference P for all groups with the same value of A separately. It is also extended to NULL values: For P grouping A a group with A = NULL is also considered. To avoid this, P grouping$_{\neg N}$ A is the grouping preference, where a NULL-group is only considered if no other values for A exist.

5 NULL Values in Preference SQL

While previous sections describe a formal framework for NULL handling in preference queries, we now present the extension of the Preference SQL query language. First, we summarize basic features of Preference SQL before describing the extended syntax. Finally, a use case scenario illustrates the applicability of the novel approach.

5.1 Preference SQL

Preference SQL [12] is a declarative extension of standard SQL by strict partial order preferences, behaving like soft constraints under the BMO query model. The BMO-set as result of a preference query contains all database tuples which are not dominated by any other tuple concerning the users preferences, cp. [10]. Syntactically, Preference SQL extends the SELECT statement of SQL by an optional *PREFERRING* clause leading to the following schematic design:

```
SELECT       ...  <selection>
FROM         ...  <table_reference>
WHERE        ...  <hard_conditions>
 PREFERRING  ...  <soft_conditions>
 GROUPING    ...  <attribute_list>
 BUT ONLY    ...  <but_only_condition>
 TOP         ...  <number>
GROUP BY     ...  <attribute_list>
HAVING       ...  <hard_conditions>
ORDER BY     ...  <attribute_list>
LIMIT        ...  <number>
```

The keywords SELECT, FROM, WHERE, GROUP BY, HAVING, and ORDER BY are treated as standard SQL keywords. The *PREFERRING* clause specifies a preference by means of the preference constructors given in Section 3. Furthermore, the Pareto preference can be expressed using the *AND* keyword in the *PREFERRING* clause, *PRIOR TO* expresses a Prioritization. Keywords such as *GROUPING* are provided to modify preference evaluation, *BUT ONLY* for the definition of post-filter or *TOP* and *LIMIT* to regulate the number of results.

A specified preference is evaluated on the result of the hard conditions stated in the WHERE clause. Therefore, preference queries can be cleanly composed with standard SQL queries, even if the standard SQL handling of NULL values uses a three-valued logic in contrast to the two-valued boolean logic used in our preference queries.

***Example* 7.** The preferences $P_1 \otimes P_2$ from Example 4 can be expressed in Preference SQL as follows:

```
SELECT ID FROM tours
PREFERRING length AROUND 4, 2
 AND vista IN ('excellent', 'good');
```

5.2 Extended Preference SQL Syntax

Following the formal framework presented in Section 4, the Preference SQL syntax has to be intuitively extended to allow the expression of newly defined NULL handling possibilities.

For NULL-insertion into the layers of categorical preferences this is straight-forward, as shown in the following example:

***Example* 8.** We translate Example 5 into Preference SQL:

```
... PREFERRING difficulty LAYERED
    (('easy'), ('medium'),
     ('hard', NULL), OTHERS)
```

For the other placements of NULL the syntax

[Attribute] [Constructor] [Parameter] [NULL-handling]

is used. The first three parts of the term are interpreted as usual, say that they are formally $P = C_d(A, X, \cong_P)$. If the optional [NULL-handling] term is set, then a preference $P = C_d(A, X, \cong_P, N)$ is constructed, where N is assigned as follows:

- **`AVOID NULL`**: NULL becomes least preferred, i.e. $N = \mathrm{N}_{\infty}^{\mathrm{dist}}$.
- **`WITH NULL AT BMO`**: NULL is incomparable, i.e. $N = \mathrm{N}_{?}$. Note that an incomparable NULL implies that NULL always occurs in the BMO-set, because incomparable values cannot be dominated by any other value.
- **`WITH NULL AT DISTANCE`** v: NULL is placed at distance v from optimal value, i.e. $N = \mathrm{N}_{v}^{\mathrm{dist}}$.
- **`WITH NULL AT LEVEL`** v: NULL is placed at level v, i.e. $N = \mathrm{N}_{v}^{\mathrm{level}}$.
- **`WITH NULL WORST`**: NULL is placed at the same distance as the worst value of $\mathrm{dom}(A)$, i.e. $N = \mathrm{N}_{\max}^{\mathrm{dist}}$.

If the [NULL-handling] term is omitted, the placement $N = \mathrm{N}_{\max}^{\mathrm{dist}}$ is used, i.e. **`WITH NULL WORST`** is the default NULL-handling.

To avoid the NULL-group in the grouping preference, i.e. to use "P grouping$_{\neg N}$ A", the syntax `[P] GROUPING [A]` **`AVOID NULL`** is used. Then a NULL-group is only considered if no other values for A exist.

5.3 Use Case

Each of the presented NULL handling strategies can be assigned to a user type. Given the database relation in Table 1 we can define four different types of user:

- **experienced user:** Sue is an experienced tour guide, knowing a lot of tours by heart. Hence, she wants to substitute unknown values with concrete values from her experience.
- **indifferent user:** Bob is quite spontaneous and doesn't care about the functionality of database systems. He knows nothing about unknown values and just wants to get best matching tours with no strings attached.
- **cautious user:** Mark is a cautious and accurate person who plans all tours in detail. He prefers tours that give him all the information to make a conscious decision. Thus, unknown values are the last thing that he wants.
- **adventurous user:** Tina is adventurous and ready to tackle new challenges. She likes to get surprised by new tours and to correct missing values with her own hiking records. Hence, she prefers tours with unknown values over fully documented tours.

Given the extended Preference SQL syntax, all these users are now able to express their individual opinions concerning NULL values.

***Example* 9.** Sue as experienced user knows that the average tour length in the desired area is about 35 kilometers and that tours with unknown difficulty level are rarely difficult tours. Since she generally prefers tours with a length around 50 kilometers and a hard difficulty level she poses the following Preference SQL query:

```
SELECT * FROM tours PREFERRING
 length AROUND 50 WITH NULL AT DISTANCE 15
 AND
 difficulty IN ('hard') with NULL AT LEVEL 1;
```

Sue specified an explicit distance value that should be used for comparisons with NULL. Additionally, she placed NULL at level 1 of a POS-preference, thus putting it into the same level as *easy* and *medium*. As result the following tuples are returned from Table 1:

id	length	difficulty	rating	vista
2	NULL	easy	5	bad
3	NULL	NULL	2	bad
4	13.1	hard	2	good

Because NULL is put at distance 15, thus equally preferred as the the value 50 – 15 = 35 for the length attribute, the tuples with id 2 and 3 are best matches w.r.t. the stated AROUND preference. Furthermore, NULL is in the same level as *easy*, hence both tuples are retrieved. Additionally, the tuple with id 4 best matches the preference for tours of difficulty *hard*. Consequently, tuples with NULL values can be part of the BMO-set in Sue's case.

***Example* 10.** Bob as indifferent user prefers tours with excellent vista and lowest tour length:

```
SELECT * FROM tours PREFERRING
 vista IN ('excellent') AND length LOWEST;
```

Since Bob doesn't know much about NULL values, he posed a query without explicit NULL handling, hence the default behavior is in place. Here, NULL values are treated as being equally preferable to the worst known attribute values, similar to *NULL WORST*. As result the following tuples are returned from Table 1:

id	length	difficulty	rating	vista
5	7.3	NULL	1	excellent

The tuple with id 5 is a best match considering the POS preference and has the lowest length of all tours. For other preferences terms, NULL values might still occur but less frequently compared to Example 9 or 12.

***Example* 11.** Mark as cautious user is looking for tours of difficulty *very easy* and thus poses the following Preference SQL query:

```
SELECT * FROM tours PREFERRING
 difficulty IN ('very easy') AVOID NULL;
```

Mark choses to avoid NULL values in the difficulty attribute if possible, hence NULL is treated as worse than the worst known attribute values. As result the following tuples are returned from Table 1:

id	length	difficulty	rating	vista
1	23.5	medium	4	excellent
2	NULL	easy	5	bad
4	13.1	hard	2	good

Mark didn't get any tuples containing NULL values in the difficulty attribute. Even in the absence of perfect matches for his preference, the best alternatives that are not of value NULL are returned.

***Example* 12.** Tina as adventurous user likes tours with a length between 15 and 20 kilometers with a tolerance of 5 kilometers. With a lower priority she is also interested in a medium difficulty:

```
SELECT * FROM tours PREFERRING
 length BETWEEN 15,20,5 WITH NULL AT BMO
 PRIOR TO
 difficulty IN ('medium') WITH NULL AT BMO;
```

She specifies NULL to be a best match for each base preference. As result the following tuples are returned from Table 1:

id	length	difficulty	rating	vista
1	23.5	medium	4	excellent
3	NULL	NULL	2	bad

Without having the NULL-handling parameter Tina would get the tuple 1. Since she is adventurous the tuple 4 having NULLs in both attributes is also returned. She may decide now.

The presented examples illustrate that the different possibilities of treating NULL values in Preference SQL have a direct impact on the returned BMO-sets. In contrast to hard constraints, none of the presented possibilities for NULL handling can guarantee that no NULL values enter the BMO-set. Even by selecting "AVOID NULL" as handling strategy, complex preference queries might still return a BMO-set containing NULLs, e.g. if a tuple with NULL in one dimension is a perfect match considering another dimension in a Skyline query.

6 Summary and Outlook

In this workshop paper we have addressed the problem of preference database queries over incomplete data, i.e., data having NULL values. We introduced a NULL handling which extends preference algebra and can easily be integrated in preference query languages. We have proposed an insertion strategy for NULL in common preference constructors and extended the syntax of Preference SQL to handle incomplete data. In contrast to other approaches for incomplete data, the transitivity relation among data tuples is preserved, thus all existing techniques for preference or Skyline query evaluation are still applicable. However, we observed that some preference optimization laws [3, 8] – independent of our NULL handling approach – cannot be applied if NULL values exist in a database relation. Although we proposed a model for NULL handling, its benefit must be evaluated in an practical use-case. For this we will extend Preference SQL with our NULL handling behavior and will do some case-studies. Of course, our approach for NULL handling is not restricted to database queries. It can also be adopted by other preference models, e.g., in the wide area of artificial intelligence and social choice theory.

REFERENCES

[1] S. Börzsönyi, D. Kossmann, and K. Stocker, 'The Skyline Operator', in *Proceedings of the 17th ICDE*, pp. 421–430, Washington, DC, USA, (2001). IEEE Computer Society.

[2] C.-Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang, 'Finding k-Dominant Skylines in High Dimensional Space', in *Proceedings of the 2006 ACM SIGMOD*, pp. 503–514, New York, NY, USA, (2006). ACM.

[3] J. Chomicki, 'Preference Formulas in Relational Queries', in *ACM TODS*, volume 28, pp. 427–466, New York, NY, USA, (2003). ACM Press.

[4] J. Chomicki, 'Logical Foundations of Preference Queries', *IEEE Data Eng. Bull.*, **34**(2), 3–10, (2011).

[5] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, 'Skyline with Presorting', in *Proceedings of the 19th ICDE*, pp. 717–816, (2003).

[6] E. Franconi and S. Tessaris, 'On the Logic of SQL Nulls.', in *Proceedings of the 6th AMW on Foundations of Data Management*, volume 866 of *CEUR Workshop Proceedings*, pp. 114–128. CEUR-WS.org, (2012).

[7] P. Godfrey, R. Shipley, and J. Gryz, 'Maximal Vector Computation in Large Data Sets', in *Proceedings of the 31st VLDB*, pp. 229–240. VLDB Endowment, (2005).

[8] B. Hafenrichter and W. Kießling, 'Optimization of Relational Preference Queries', in *Proceedings of the 16th ADC*, pp. 175–184, Darlinghurst, Australia, (2005). Australian Computer Society, Inc.

[9] M. E. Khalefa, M. F. Mokbel, and J. J. Levandoski, 'Skyline Query Processing for Incomplete Data', in *Proceedings of the 2008 IEEE 24th ICDE*, pp. 556–565, Washington, DC, USA, (2008). IEEE Computer Society.

[10] W. Kießling, 'Foundations of Preferences in Database Systems', in *Proceedings of the 28th VLDB*, pp. 311–322, Hong Kong, China, (2002). VLDB Endowment.

[11] W. Kießling, 'Preference Queries with SV-Semantics', in *Proceedings of the 11th COMAD*, eds., Jayant R. Haritsa and T. M. Vijayaraman, pp. 15–26, Goa, India, (2005). Computer Society of India.

[12] W. Kießling, M. Endres, and F. Wenzel, 'The Preference SQL System - An Overview', *Bulletin of the Technical Commitee on Data Engineering, IEEE Computer Society*, **34**(2), 11–18, (2011).

[13] W. Kießling, M. Soutschek, A. Huhn, P. Roocks, M. Endres, S. Mandl, F. Wenzel, and A. Zelend, 'Context-Aware Preference Search for Outdoor Activity Platforms', Technical Report 2011-15, University of Augsburg, (2011).

[14] D. Papadias, Y. Tao, G. Fu, and B. Seeger, 'Progressive Skyline Computation in Database Systems', *ACM Trans. Database Syst.*, **30**(1), 41–82, (2005).

[15] J. Pei, B. Jiang, X. Lin, and Y. Yuan, 'Probabilistic Skylines on Uncertain Data', in *VLDB*, pp. 15–26. ACM, (2007).

[16] T. Preisinger and W. Kießling, 'The Hexagon Algorithm for Evaluating Pareto Preference Queries', in *Proceedings of the 3rd MPref*, (2007).

[17] Francesca Rossi, Brent Venable, and Toby Walsh, *A Short Introduction to Preferences: Between Artificial Intelligence and Social Choice*, volume n/a of *Synthesis Lectures on Artificial Intelligence and Machine Learning*, Morgan & Claypool, July 2011.

[18] K. Stefanidis, G. Koutrika, and E. Pitoura, 'A Survey on Representation, Composition and Application of Preferences in Database Systems', *ACM TODS*, **36**(4), (2011).

[19] F. Wenzel, M. Soutschek, and W. Kießling, 'A Preference SQL Approach to Improve Context-Adaptive Location-Based Services for Outdoor Activities', in *Advances in Location-Based Services*, 191–207, Springer Berlin Heidelberg, (2011).