# The Preference SQL System – An Overview

Werner Kießling, Markus Endres, Florian Wenzel
University of Augsburg, Institute for Computer Science
86135 Augsburg, Germany
{*kiessling, endres, wenzel*}@*informatik.uni-augsburg.de*

**Abstract**

*Preference SQL is a declarative extension of standard SQL by strict partial order preferences, behaving like soft constraints under the BMO query model. Preference queries can be formulated intuitively following an inductive constructor-based approach. Both qualitative methods like e.g. Pareto / skyline and quantative methods like numerical ranking, definable over categorical as well as numerical attribute domains can be used. The Preference SQL System is implemented as a middleware component, enabling a seamless application integration with standard SQL back-end systems. The preference query optimizer performs algebraic transformations of preference relational algebra as well as cost-based algorithm selection e.g. for efficient Pareto / skyline evaluation. Ongoing work extends Preference SQL towards efficient support for personalized location-based mobile geo-services and social networks.*

## 1 Introduction

After years of intensive experiences in particular through the Internet, search engine technology has improved considerably. However, more often than desirable it still does not meet the user's expectations. Prominent deficiencies which probably all search engine users have suffered are the infamous "empty-result" effect (i.e., getting no query results) and its opposite companion the notorious "flooding effect" (i.e., getting far too many query results). Thus a typical user search session requires several trial-and-error style query attempts, until eventually something suitable has been found, or the user quits frustratedly. Ad hoc attempts to remedy this unpleasant situation such as parametric search can achieve some relief, but do not solve the intrinsic problems. Focusing on search engines based on SQL databases, the cause for these difficulties is easily identified: SQL only supports hard constraints with an exact-match semantics. In contrast, mostly people also include preferences in their decisions, both in their daily lives as well as in business. But the very nature of preferences is different from hard constraints. Preferences are like soft constraints, requiring a match-making process instead: If my favorite choice is available in the database, I will take it. Otherwise, instead of getting nothing, I am open to alternatives, but show me only the best ones available in the database. Therefore, improving SQL-based search capabilities asks for extending SQL query technology towards a preference model, which of course should be powerful, flexible and intuitive for the user, but simultaneously query performance must be fast enough. However, the question of "what is the right preference model" is far from being trivial. In fact, the complexity of preferences has challenged researchers from diverse disciplines. During the past fifty years preferences have traditionally

**Bulletin of the IEEE Computer Society Technical Committee on Data Engineering**

been studied in fields such as Economic Decision Making, Social Choice Theory, Operations Research and Artificial Intelligence (see [3] for an overview). In recent years preferences have found steadily increasing interest in databases as well. A survey on the state of the art in databases can be found in [15].

In this paper we adopt the preference model introduced by [8], suggesting preferences to be strict partial orders (SPO). A similar approach has been proposed by [4]. SPO preferences can be interpreted in a very intuitive manner as personalized wishes in the form of "I like A more than B". Formally, a *preference P* on a set of attributes $A$ is defined as $P := (A, <_P)$, where $<_P \subseteq dom(A) \times dom(A)$ is a strict partial order. $x <_P y$ is interpreted as "I like y more than x". Note that the preference order $<_P$ is irreflexive and transitive. An important subclass of preferences are *weak order preferences* (WOP), i.e., SPOs for which negative transitivity holds: $\forall x, y, z \in dom(A)$: $\neg(x <_P y) \wedge \neg(y <_P z) \Rightarrow \neg(x <_P z)$. For a WOP $P = (A, <_P)$ the dominance test can be efficiently performed by a numerical *utility function* $f : dom(A) \to \mathbb{R}_0^+$ which depends on the type of preference. Dominated tuples have higher function values, i.e., $x <_P y \iff f(x) > f(y)$. Note that this preference definition does not explicitly take the user's situation or context into account (see [12] for a discussion on context-awareness). Instead, preference-driven applications are supposed to maintain a persistent preference repository in order to automatically retrieve proper preferences for context-dependent query composition (see [7] for details).

Based on this preference model a first version of Preference SQL, extending SQL queries by preferences, has been described in [12]. An early commercial implementation of Preference SQL was available already in 1999. For the sake of rapid prototyping and quick time to market this version implemented loose coupling to a standard SQL database together with a query re-writing approach. Since then state of the art for preference handling in database systems has advanced considerably (see [15]). Here we provide an overview on the recent version of Preference SQL developed at the University of Augsburg.

## 2 The Preference Query Model

### 2.1 Preference Constructors

For the design of the declarative preference query language in [8] an inductive, constructor-based approach was used to formally specify a preference. For this purpose, a choice of intuitive *base preference constructors* together with some *complex preference constructors* has been defined, all being SPOs.

**Base Preferences** Preferences on single attributes are called *Base preferences*. There are base preference constructors for *continuous* and for *discrete (categorical)* domains. Figure 1 shows the taxonomic "ISA"-hierarchy of several frequently occuring base preference constructors.

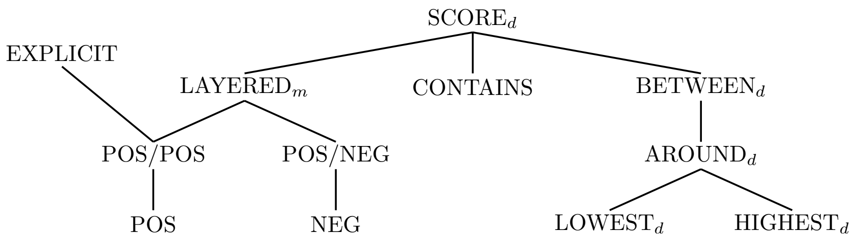

Figure 1: Taxonomy of base preference constructors

**Definition 1 (SCORE$_d$ Preference):** Given a scoring function $f_d : dom(A) \to \mathbb{R}$, and some $d > 0$. Then $P$ is called a SCORE$_d$ preference, iff for $x, y \in dom(A)$: $x <_P y \iff f_d(x) > f_d(y)$ where $f_d(v), v \in dom(A)$, is defined as: $f_d(v) = \lceil f(v)/d \rceil$. Note that $f(v)$ is a numerical utility function.

When dealing with numerical values, it is a common practice to group ranges of scores together. Such a real-world behavior can be modeled using a $d$-parameter. It maps different function values to a single number which therefore maybe become substitutable with other values.

Since SCORE$_d$ specifies a WOP, all sub-constructors of SCORE$_d$ in Figure 1 are WOPs, too. However, EXPLICIT is not a WOP, hence not a sub-constructor of SCORE$_d$. EXPLICIT is used to create any preference that can be expressed by a finite set of "A is better than B" relationships. The continuous preference constructor BETWEEN$_d$ expresses the wish for a value between a *lower* and an *upper* bound.

**Definition 2 (BETWEEN$_d$ Preference):** The BETWEEN$_d$(A, [low, up]) preference is a SCORE$_d$ preference with the following scoring function for $low, up \in dom(A)$:

$$f(v) := \begin{cases} low - v & ,\text{if} \quad v < low \\ 0 & ,\text{if} \quad low \leq v \leq up \\ v - up & ,\text{if} \quad up < v \end{cases}$$

Specifying $low = up$ $(=: z)$ in BETWEEN$_d$ we get the AROUND$_d$(A, z) preference constructor. In the AROUND$_d$(A, z) preference the desired value should be $z$. If this is infeasible, values within a distance of $d$ are acceptable. Furthermore, specializing $z = \inf_A$ and $z = \sup_A$ yield the constructors LOWEST$_d$(A) and HIGHEST$_d$(A), resp., where $\inf_A$ / $\sup_A$ are the infimum / supremum of $dom(A)$. HIGHEST$_d$ and LOWEST$_d$ allow users to express easily their desire for values as high or as low as possible.

**Definition 3 (LAYERED$_m$ Preference):** Let $L = (L_1, \cdots, L_{m+1})$ $(m \geq 0)$ be an ordered list of $m+1$ sets forming a partition of $dom(A)$ for an attribute $A$. The preference $P$ is a LAYERED$_m$ preference if it is a SCORE$_0$ preference with the following scoring function: $f(x) := i - 1 \iff x \in L_i$. For convenience, one of the $L_i$ may be named "others", representing the set $dom(A)$ without the elements of the other subsets.

All categorical base preferences are sub-constructors of LAYERED$_m$, e.g. POS(A, POS-set) is equal to LAYERED$_1$(A, POS-set, others), and 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. In addition to the discussed preferences we support the base preference CONTAINS which works on text attributes and currently supports simple full-text search. For details on all preferences we refer to [8, 10].

**Complex Preferences** If one wants to combine several preferences into an overall preference, 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". We model equal importance of preferences by the so-called Pareto preference, whereas for ordered importance we introduce prioritization. A generalization to more than two preferences is straightforward, cp. [10].

**Definition 4 (Pareto Preference):** Assume preferences $P_1 = (A_1, <_{P_1})$, $P_2 = (A_2, <_{P_2})$, and $x = (x_1, x_2)$, $y = (y_1, y_2) \in dom(A)$, then the Pareto preference constructor denoted by $P := P_1 \otimes P_2$ is defined as:

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

**Definition 5 (Prioritization):** Assume two 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: $(x_1, x_2) <_P (y_1, y_2)$ iff $x_1 <_P y_1 \vee (x_1 = y_1 \wedge x_2 <_{P_2} y_2)$.

Generalizing above definitions for Pareto and prioritization to capture the semantics of "equally good" without violating the SPO property is a non-trivial issue. The *substitutable values* (SV) semantics [10] provides the most general way to achieve this goal by characterizing, which domain values can be considered as "equally good", hence being "substitutable". Using *trivial* SV-semantics only *equal* values are considered as equally good (Definition 4 and Definition 5 are examples for trivial SV-semantics), whereas for *regular* SV-semantics values that are *indifferent* are considered as equally good. However, regular SV-semantics is only valid for WOPs, otherwise SPO is violated.

Another form of preference combination is by associating numerical scores to each individual preference and then applying a combining function to compute a single score, which decides the "better-than" relationship. Such *weighted importance* between preferences is realized by the numerical ranking preference, cf. [8, 10].

**Definition 6 (Numerical Ranking Preference):** Given weak order preferences $P_1, \ldots, P_m$ with scoring functions $f_1, \ldots, f_m$. Then the numerical ranking preference $\text{RANK}_{F,d}$ ($d > 0$) with an $m$-ary combining function $F : \mathbb{R}^m \rightarrow \mathbb{N}_0$ is defined as:

$$(x_1, \ldots, x_m) <_P (y_1, \ldots, y_m) \iff \left\lceil \frac{F(f_1(x_1), \ldots, f_m(x_m))}{d} \right\rceil > \left\lceil \frac{F(f_1(y_1), \ldots, f_m(y_m))}{d} \right\rceil$$

Note that $\text{RANK}_{F,d}$ yields a WOP again. On the other hand Pareto and prioritization are not WOP preserving.

## 2.2 The BMO Query Model

The declarative query model for relational databases relies on relational calculus, which is equivalent to relational algebra. Queries under this model implement the semantics of hard constraints. All retrieved elements exactly satisfy the specified constraints. Therefore this model is prone to the empty-result effect mentioned earlier. When trying to overcome it by applying tedious parametric search, Boolean expert search or similar ad-hoc remedies, often the opposite effect of flooding the user with too many, mostly irrelevant results occurs. All of this can be avoided by supporting the BMO query model for preferences as follows.

**Definition 7 (Preference Selection, BMO-set):** Let $P = (A, <_P)$ be a preference and $R(A_1, \cdots, A_m)$ be a database relation with $A \subseteq \{A_1, \cdots, A_m\}$. Then the preference selection operator is defined by:

$$\sigma[P](R) := \{t \in R \mid \neg\exists t' \in R : t[A] <_P t'[A]\}$$

The result of preference selection (also named "winnow"-operator in [4]) is called BMO-set.

Preference selection is defined by means of negations, hence it defines a form of non-monotonic reasoning. It 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 off worse results. Thus preferences are treated as *soft* constraints. Note that Pareto preference queries form a superset of the skyline operator [2]. However, the skyline operator only allows LOWEST and HIGHEST preferences. In contrast, a Pareto preference can be constructed from any preferences presented in Section 2.

## 3 The Preference SQL Query Language

Before illustrating the syntax of Preference SQL we shortly address the formal semantics of Preference SQL, which extends standard SQL with its exact match semantics by SPO preferences under the BMO query model. The declarative reasoning paradigm of SQL for relational databases is founded on the familiar model theory based on relational calculus, its operational equivalent being relational algebra. It is known that this can be extended towards recursive relational databases. For preference queries with BMO semantics, subsumption models [13] can serve as a theoretical foundation for the declarative semantics, with an equivalent operational semantics expressed by the preference selection operator. Our current version of Preference SQL is an extension to SQL-92 compliant (i.e., non-recursive) relational databases.

Syntactically, Preference SQL [12, 6] extends the SELECT statement of SQL by an optional PREFERRING clause. A preference query selects all interesting tuples, i.e., tuples that are not dominated by other tuples. Preference SQL currently supports most of the SQL-92 standard as well as all previously mentioned base preferences together with the constructors for Pareto, prioritization and Rank. A Preference SQL query block has the following schematic design:

```
SELECT          ...  <selection>
FROM            ...  <table _reference>
WHERE           ...  <hard _conditions>
 PREFERRING     ...  <soft _conditions>
 GROUPING       ...  <attribute  _list>
 BUT ONLY       ...  <but _only _condition>
GROUP BY        ...  <attribute  _list>
HAVING          ...  <hard _conditions>
ORDER BY        ...  <attribute  _list>
```

Figure 2: Preference SQL query block.

The keywords SELECT, FROM, WHERE, GROUP BY, HAVING, and ORDER BY are treated as the standard SQL query keywords. The other keywords are explained subsequently. For further syntax details of Preference SQL we refer to [1] and [12].

▷ `PREFERRING`: The PREFERRING clause specifies a (possibly very complex) preference $P$ by means of the preference constructors stated in Section 2. $P$ is evaluated on the result $T$ of the hard conditions stated in the WHERE clause, returning the BMO-set of $P$. Note that in this way Preference SQL implements a generalization of **constrained skyline queries** [16, 5]. Empty result sets can only occur when the WHERE clause returns an empty result $T$. Note that all WOPs can be specified with either *trivial* or *regular* SV-semantics omitting or using the keyword REGULAR after the preference definition.

▷ `GROUPING`: Instead of evaluating $P$ on the entire intermediate relation $T$, the GROUPING clause partitions $T$ w.r.t. the given attribute list. Then P is evaluated on each partition of $T$ separately.

▷ `BUT ONLY`: There are cases, where the BMO-set returned by the PREFERRING / GROUPING clause is too large for a given application scenario. Note that his tends to hold in particular for high-dimensional Pareto / skyline queries. Then within the BUT ONLY clause one can specifiy hard conditions similar to the WHERE clause. BUT ONLY is evaluated on the result of PREFERRING / GROUPING, hence implementing a generalization of **skyline queries with constraints**, cp. [16].

**Example 1:** We want to find all cars with their owners and address who live in Augsburg or Munich. The average fuel consumption on highway and city driving must be equal or less than 6 liter / 100 km. The cars should have a mileage between 40.000 km and 50.000 km, and a price around 12.000 Euro. A difference of 2.000 Euro does not matter. This Pareto preference is more important than the make being a BMW or an Audi. The full-text description of the car should contain "ABS brake". This is equally important to the make. The preference is combined with a RANK preference on the age and the price of the car. Furthermore, only cars with a horsepower less than 100 hp should be presented after preference evaluation (BUT ONLY). The result has to be ordered by the registration date of the cars. This Preference SQL query is depicted in Figure 3.

```
SELECT o.id, a.id, c.id
FROM owner o, address a, car c
WHERE   a.city = 'Augsburg' OR a.city = 'Munich' AND
         a.id = o.idaddress AND o.id = c.idowner  AND
         (c.city_consumption + c.highway_consumption) / 2 <= 6
 PREFERRING  (c.mileage BETWEEN (40000 AND 50000) AND
                  c.price AROUND 12000, 2000)                        PRIOR TO
               (c. price HIGHEST AND c.make IN ('BMW', 'Audi') REGULAR  AND
                  c.description CONTAINS 'ABS brake'                 AND
               (c.age SCORE 'age_function', c.   price SCORE 'price_function')
                  RANK age_price_function )
 BUT ONLY c.hp < 100 ORDER BY c.reg_date;
```

Figure 3: A sample complex Preference SQL query.

The WHERE clause states the join conditions. In Preference SQL syntax, Pareto preferences are stated as "AND" in the PREFERRING clause, while prioritization is expressed by the keyword "PRIOR TO". The keywords "IN" represents a POS preference, "CONTAINS" expresses the full-text search. The RANK preference consists of score functions which must be implemented in Preference SQL itself. After the evaluation of the preferences the BUT ONLY clause filters out all result tuples which do not fulfill the horsepower condition.

Note that it is possible to specify several preferences on the same attribute. In the query above there are three preferences on the price of the car, which might for example come from different persons. We call such preferences *social group preferences* because one important application is to use them to model group decisions and group formations depending on individual group member preferences.

## 4 Preference Query Evaluation

### 4.1 The Preference SQL System Architecture

Extending existing SQL database systems towards Preference SQL requires some crucial design decisions. The current University version of Preference SQL adopts a (Java 1.6-based) middleware approach as depicted in Figure 4.

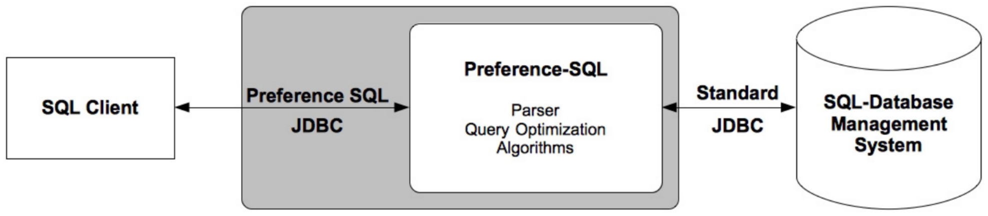

Figure 4: System architecture of Preference SQL

For a seamless application integration we have extended standard JDBC technology towards a Preference SQL / JDBC package. This enables the client application to submit Preference SQL queries through familiar SQL clients. The Preference SQL middleware parses the query and performs preference query optimization as described subsequently. Those parts of an optimized query execution plan, which directly correspond to SQL-92, are retranslated into SQL-92 and handed over to the attached SQL database system for evaluation. The new preference selection operator together with novel algebraic and cost-based query optimization methods are dealt with in the Preference SQL middleware. This approach has proven its flexibility and performance in various prototype applications conducted so far, see [11] for example.

### 4.2 Preference Query Optimization

Being an extension of standard SQL, Preference SQL can inherit the accumulated vast query optimization knowhow from relational databases. Moreover, additional optimization techniques are required to cope with the challenges posted by the preference selection operator for complex strict partial order preferences. Since Preference SQL always works with a standard SQL database system as back-end, another design goal for the preference query optimizer has been to offload parts of the query evaluation work to this back-end. Basically, in its current version a Preference SQL query is processed as follows:

1) The query gets parsed and transformed into an initial operator tree for preference relational algebra, which comprises classical relational algebra plus the preference selection operator.
2) Then heuristics for algebraic operator tree transformation are applied. For this purpose, the familiar principles known from standard relational databases had to be augmented by new transformation laws for preference relational algebra (see [6, 4, 5] for details).
3) For efficient evaluation of the preference selection operator in our middleware, special efficient algorithms had to be implemented. For instance, for Pareto / skyline queries this repertoire includes the Hexagon algorithm [14] and LESS [8]. The algorithm selection is guided by a cost-based query execution model.
4) Likewise guided by a cost-based model, the Preference query optimizer determines suitable sub-trees of the final optimized operator tree for offloading them to the back-end SQL system. Those sub-trees are re-translated into SQL and sent via JDBC to the attached back-end afterwards.
5) The entire result of the Preference SQL query is assembled in our middleware and returned to the requesting client by Preference SQL JDBC.

It has to be pointed out that many optimization methods applied above would become invalid if transitivity of preferences would get violated.

## 5 Summary and Outlook

We have presented an overview on the Preference SQL system, which extends standard SQL by preferences. Preference evaluation follows the BMO paradigm, treating preferences as soft constraints. For ease of use a variety of preference constructors on categorical as well as on numerical domains are available. Since preferences are always strict partial orders, by means of composition complex preferences can be formulated inductively. This includes Pareto preferences, prioritization and also numerical ranking. Seamless application integration is supported by a Preference SQL extension to JDBC, enabling easy access to standard SQL systems like, e.g., Oracle, MySQL or PostgresSQL. By this means, standard SQL clients like e.g. Squirrel or DBVisualizer, or even mobile iPhone or Android clients, can execute Preference SQL queries immediately. Our preference query optimization implements algebraic methods as well as cost-based approaches, including specialized evaluation methods like, e.g., the Hexagon algorithm for Pareto queries. A demo of Preference SQL is available at [1].

An enhancement is concerning a new Top-k interface for preference queries on top of the already established BMO evaluation. Also we are planning to advance existing text search capabilities to a more powerful preference based full-text search. Finally, the use of ontologies in combination with preferences is of interest. Besides these changes to the core functionality, current research efforts target the special support of Preference SQL for innovative application areas such as location-based services, mobile geo-services and social networking sites. Hence, the development of preference constructors on geospatial data that is e.g. stored in a PostGIS system as well as possibilities to extend context-awareness are ongoing research. Personalized routing problems will also ask for extensions towards recursive Preference SQL. In social networks, aspects of social matching, social group preferences and group recommendations are driving further developments of Preference SQL.

## References

[1] The Preference SQL Syntax and Trial Version. *http://www.trial.preferencesql.com/*

[2] S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In Proceedings of the 17th International Conference on Data Engineering (ICDE), pp. 421–430, Washington, USA, 2001.

[3] R. I. Brafman, and C. Domshlak. Preference Handling: An Introductory Tutorial. In AI Magazine 30 (1), pp. 58 - 86, Menlo Park, CA, USA, 2009.

[4] J. Chomicki. Preference Formulas in Relational Queries. In ACM Transactions on Database Systems (TODS), volume 28, pp. 427–466, New York, NY, USA, 2003.

[5] M. Endres and W. Kießling. Semi-Skyline Optimization of Constrained Skyline Queries. In Proceedings of the 22nd Australasian Database Conference (ADC), Perth, Australia, 2011.

[6] B. Hafenrichter and W. Kießling. Optimization of Relational Preference Queries. In Proceedings of the 16th Australasian Database Conference (ADC), pp. 175–184, Darlinghurst, Australia, 2005.

[7] S. Holland, and W. Kießling. Situated Preferences and Preference Repositories for Personalized Database Applications In Proceedings of the 23rd International Conference on Conceptual Modeling (ER), pp. 511–523, Shanghai, China, 2004.

[8] P. Godfrey, R. Shipley, and J. Gryz. Maximal Vector Computation in Large Data Sets. In Proceedings of the 31st International Conference on Very Large Data Bases (VLDB), pp. 229–240, Trondheim, Norway, 2005.

[9] W. Kießling. Foundations of Preferences in Database Systems. In Proceedings of the 28th International Conference on Very Large Data Bases (VLDB), pp. 311–322, Hong Kong, China, 2002.

[10] W. Kießling. Preference Queries with SV-Semantics. In Proceedings of the 11th International Conference on Management of Data (COMAD), pp. 15–26, Goa, India, 2005.

[11] W. Kießling, S. Fischer, and S. Döring. COSIMA$^{B2B}$ - Sales Automation for E-Procurement. In The 6th IEEE Conference on E-Commerce Technology (CEC), pp. 59–68, Los Alamitos, CA, USA, 2004.

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

[13] G. Köstler, W. Kießling, H. Thöne, and U. Güntzer. Fixpoint Iteration with Subsumption in Deductive Databases. In Journal of Intelligent Information Systems, Vol. 4, pp. 123–148, 1995.

[14] T. Preisinger and W. Kießling. The Hexagon Algorithm for Evaluating Pareto Preference Queries. In Proceedings of the 3rd Multidisciplinary Workshop on Advances in Preference Handling (M-PREF in conjunction with VLDB '07), Vienna, Austria, 2007.

[15] K. Stefanidis, G. Koutrika, and E. Pitoura. A Survey on Representation, Composition and Application of Preferences in Database Systems. In ACM Transactions on Database Systems (TODS), To appear in Vol. 36, Issue 4, New York, NY, USA, 2011.

[16] M. Zhang, and R. Alhaji. Skyline queries with constraints: Integrating skyline and traditional query operators. In Data & Knowledge Engineering, pp. 153–168, Elsevier, 2010.