A Survey on Representation, Composition and Application of Preferences in Database Systems

KOSTAS STEFANIDIS
Chinese University of Hong Kong, Hong Kong
GEORGIA KOUTRIKA
IBM Almaden Research Center, USA
and
EVAGGELIA PITOURA
University of Ioannina, Greece

Preferences have been traditionally studied in philosophy, psychology and economics and applied to decision making problems. Recently, they have attracted the attention of researchers in other fields, such as databases where they capture soft criteria for queries. Databases bring a whole fresh perspective to the study of preferences, both computational and representational. From a representational perspective, the central question is how we can effectively represent preferences and incorporate them in database querying. From a computational perspective, we can look at how we can efficiently process preferences in the context of database queries. Several approaches have been proposed but a systematic study of these works is missing. The purpose of this survey is to provide a framework for placing existing works in perspective and highlight critical open challenges to serve as a springboard for researchers in database systems. We organize our study around three axes: preference representation, preference composition and preference query processing.

Categories and Subject Descriptors: H.2.4 [**Database Management**]: Systems—*Relational databases*

General Terms: Algorithms, Design, Languages

Additional Key Words and Phrases: preference modeling, preference queries

1. INTRODUCTION

Preferences guide human decision making from early childhood (e.g., "which ice cream flavor do you prefer?") up to complex professional and organizational decisions (e.g., "which investment funds to choose?"). Preferences have traditionally been studied in philosophy, psychology and economics and applied to decision

Authors' addresses: K. Stefanidis, Department of Computer Science and Engineering, Chinese University of Hong Kong, Hong Kong; email: kstef@cse.cuhk.edu.hk. G. Koutrika, IBM Almaden Research Center, USA; email: gkoutri@us.ibm.com. E. Pitoura, Department of Computer Science, University of Ioannina, Ioannina, Greece; email: pitoura@cs.uoi.gr.

making problems. For instance, in philosophy, they are used to reason about values and desires [Hansson 2001]. In mathematical decision theory, preferences (or utilities) model economic behavior [Fishburn 1999]. The notion of preference has in recent years drawn new attention from researchers in other fields, such as artificial intelligence, where they capture agents' goals [Boutilier et al. 1999; Delgrande et al. 2003; Wellman and Doyle 1991], and databases, where they capture soft criteria for database queries. Explicit preference modeling provides a declarative way to choose among alternatives, whether these are solutions of problems to solve, answers of database queries, decisions of a computational agent and so on. In this article, we focus on preferences in the context of database queries.

In databases, interest in preferences was triggered by observing the limitations of the Boolean database answer model, where query criteria are considered as *hard* by default and a non-empty answer is returned only if it satisfies all the query criteria. In this context, a user can face either of two problems: (*i*) the *empty-answer* problem, where the conditions are too restrictive or the data cannot exactly match the query or (*ii*) the *too-many-answers* problem, where too many results match the query. It is hard to cope with these problems especially if a user is not familiar with a structured query language in order to formulate accurate queries and when accessing web databases, whose schema and contents are unknown.

Incorporating soft criteria or preferences in a query can help cope with these problems. The empty-answer problem can be tackled by relaxing some of the hard constraints in the query, i.e., considering them as soft or as user wishes or by replacing them by constraints that capture preferences related to the given query and returning results that are ranked according to how well they match the modified query. The too-many-answers problem can be tackled by strengthening the query with additional preferences to rank and possibly focus the query results.

The study of preference queries in databases originated by Lacroix and Lavency [1987], who proposed a simple extension of the relational calculus in which preferences for tuples satisfying given logical conditions can be expressed. For instance, one could say: pick the tuples of R satisfying $Q \wedge P1 \wedge P2$; if the result is empty, pick the tuples satisfying $Q \wedge P1 \wedge \neg P2$; if the result is empty, pick the tuples satisfying $Q \wedge \neg P1 \wedge P2$. Gaasterland and Lobo [1994] introduced a simple formalism, where a user provides a lattice of domain independent values that define preferences and a set of domain specific user constraints qualified with lattice values. The constraints are automatically incorporated into a relational or deductive database through a series of syntactic transformations that produces an annotated deductive database. Query answering procedures for deductive databases are then used, with minor modifications, to obtain annotated answers to queries. Almost a decade later, the web has made information easily accessible and renewed interest in preferences was triggered by the need to make (web) databases more user-friendly.

Several approaches have been proposed since then but a systematic study of them is missing. It is the purpose of this paper to provide a framework for studying various approaches that deal with preferences in databases. In particular, our objective is to survey in a holistic way approaches that: (*i*) define preferences, (*ii*) combine preferences and (*iii*) apply preferences to query processing. We organize our study around these main axes as follows:

Fig. 1. Database schema.

Preference Representation: Preferences naturally come into different flavors and people may have a mix of different preferences. Works on preference modeling have focused on different aspects of the problem but two main philosophies can be distinguished on the basis of how preferences are formulated: qualitative approaches, where preferences are expressed by comparing items ("I like westerns better than comedies") and quantitative ones, where a preference for a specific item is expressed as a degree of interest in this item ("my interest in westerns is 0.8 and in comedies 0.4"). We categorize preference representation approaches using the following dimensions:

1. *Formulation*. Preferences are formulated qualitatively or quantitatively;
2. *Granularity*. Preferences can be expressed at different levels, i.e., for tuples, relations, relationships and attributes;
3. *Context*. Preferences can be context-free or can hold under specific conditions;
4. *Aspects*. Preferences may vary based on their intensity, elasticity, complexity and other aspects.

Preference Composition: Given a set of preferences over a set of tuples, different composition mechanisms can be applied to infer (e.g., implicit preferences), combine (e.g., through combining scoring functions) or override preferences (e.g., in prioritized composition) and finally, derive a ranking of the tuples on the basis of how they match these preferences. In this survey, we group preference composition mechanisms into the following categories:

1. *Qualitative composition*. These mechanisms combine preferences resulting in a relative (i.e., qualitative) ordering of the tuples;
2. *Quantitative composition*. These mechanisms combine preferences by assigning final scores to the tuples, which are thus ordered in a quantitative way;
3. *Heterogeneous composition*. These mechanisms are used to combine preferences of different granularity, for example, preferences for relationships between tuples with preferences for tuple attributes.

Preference Query Processing: Preference are used in query processing to provide users with customized results typically through ranking. There are roughly two different lines of work on using preferences in query processing. Namely, preferences are exploited through:

1. *Expanding database queries*. These methods assume the existence of a number of user preferences and appropriately rewrite regular database queries to incorporate them. This process is often referred to as *query personalization*.
2. *Employing preference operators*. These methods use special database operators (such as top-*k* or skyline) to explicitly express preferences within queries.

Our survey covers both approaches. We shall also discuss methods for improving the performance of preferential query processing, for instance, by performing off-line pre-processing steps to construct rankings of database tuples based on preferences. There is a large number of algorithms for the implementation of preference queries (especially for top-*k* and skyline queries). We do not intend to provide an exhaustive review for special classes of preference queries. We consider that drilling down to the specifics of different implementations and algorithms is

movie

	mid	title	year	director	genre	language	duration
t_1	m_1	Casablanca	1942	M.Curtiz	drama	english	102
t_2	m_2	Psycho	1960	A.Hitchcock	horror	english	109
t_3	m_3	Schindler's List	1993	S.Spielberg	drama	english	195

play

mid	aid
m_1	a_1
m_1	a_2
m_2	a_3
m_3	a_4

actor

aid	name	date_of_birth
a_1	H. Bogart	25-Dec-1899
a_2	I. Bergman	29-Aug-1915
a_3	A. Perkins	4-Apr-1932
a_4	L. Neeson	7-Jun-1952

Fig. 2. Database instance example.

the subject of separate surveys focusing on algorithms for a specific class of preference queries, such as the survey on algorithms for top-*k* queries by Ilyas et al. [2008]. Instead, we aim at providing an overview of the main approaches for different types of preference queries.

As a running example, we consider a simple database that stores information about movies, consisting of three relations: movie, play, actor. Figure 1 depicts the schema of this database. We shall also use the database instance shown in Figure 2.

This survey is organized as follows. We present existing approaches to preference representation (Section 2) followed by mechanisms for preference composition (Section 3). Then, we study preferential query processing methods (Section 4). In the final section (Section 5), we discuss other issues such as preference learning and revising, non-relational preference models, other preference applications and connections to other disciplines that deal with preferences. We conclude with a discussion on critical open challenges.

2. PREFERENCE REPRESENTATION

Understanding user preferences and finding appropriate representations for them is a real challenge. There are quite a few approaches (preference models) in the literature that deal with preference representation and composition and try to reach meaningful conclusions regarding the desired answers of a database query from different perspectives. In this section, we focus on representing individual preferences and in Section 3, on mechanisms for preference composition.

We present preference representation based on how preferences are formulated (*formulation* - Section 2.1), at what level they are expressed (*granularity* - Section 2.2), when they hold (*context* - Section 2.3) and what they express (*aspects* - Section 2.4).

2.1 Preference Formulation

In general, preferences can be expressed either qualitatively or quantitatively. In the *qualitative approach*, preferences between database tuples are specified directly, typically using binary preference relations. Preference relations may be specified using *logical formulas* [Chomicki 2003] or special *preference constructors* [Kießling 2002]. In the *quantitative approach*, preferences are expressed by assigning numerical scores to database tuples. In this case, a tuple t_i is preferred over a tuple t_j, if and only if, its score is higher than the score of t_j. Scores may be assigned through *preference functions* (e.g., [Agrawal and Wimmers 2000]) or as *degrees of interest* associated with specific conditions that must be satisfied (e.g., [Koutrika and Ioannidis 2004]).

In the following, we shall use $R(A_1, \ldots, A_d)$ to denote a relational schema with d attributes A_i, $1 \leq i \leq d$, where each attribute A_i takes values from a domain $dom(A_i)$. Let $A=\{A_1, A_2, \ldots, A_d\}$ be the attribute set of R and $dom(A) = dom(A_1) \times \ldots \times dom(A_d)$ be its value domain. We use t to denote a tuple $(u_1, u_2, \ldots, u_d) \in dom(A)$ of R and r to denote an instance (i.e., tuple set) of R. Let $B \subseteq A$ be a subset of the attribute set, $t[B]$ stands for the projection of t on B. Finally, P denotes a preference.

2.1.1 *Qualitative Preferences.* In the qualitative approach, preferences are defined as binary relations between two tuples. Given a set S, a binary relation $\mathcal{B}$ over S is a subset of the Cartesian product $S \times S$. For a pair (a, b) of $\mathcal{B}$, we use the notation $a \mathcal{B} b$, whereas, for a pair (a, b) that does not belong to $\mathcal{B}$, we use the notation $\neg(a \mathcal{B} b)$. A preference relation is defined as follows.

DEFINITION 1. *Let $R(A_1, \ldots, A_d)$ be a relational schema and $dom(A_i)$ be the domain of attribute A_i, $1 \leq i \leq d$. A preference relation $\succ_P$ over R is a subset of* $(dom(A_1) \times \ldots \times dom(A_d)) \times (dom(A_1) \times \ldots \times dom(A_d))$.

The interpretation of a preference relation $t_i \succ_P t_j$ between two tuples t_i and t_j of R is that t_i is *preferred over* t_j under $\succ_P$. We shall also say that t_i is *better than* t_j or that t_i *dominates* t_j under $\succ_P$.

Next, we list several typical properties of binary relations that are useful in classifying preference relations. A binary relation $\mathcal{B}$ over a set S is called:

—reflexive, if, $\forall a \in S$, $a \mathcal{B} a$,
—irreflexive, if, $\forall a \in S$, $\neg (a \mathcal{B} a)$,
—symmetric, if, $\forall a, b \in S$, $a \mathcal{B} b \Rightarrow b \mathcal{B} a$,
—asymmetric, if, $\forall a, b \in S$, $a \mathcal{B} b \Rightarrow \neg(b \mathcal{B} a)$,
—antisymmetric, if, $\forall a, b \in S$, $(a \mathcal{B} b \wedge b \mathcal{B} a) \Rightarrow a = b$,
—transitive, if, $\forall a, b, c \in S$, $(a \mathcal{B} b \wedge b \mathcal{B} c) \Rightarrow a \mathcal{B} c$,
—negatively transitive, if, $\forall a, b, c \in S$, $(\neg(a \mathcal{B} b) \wedge \neg(b \mathcal{B} c)) \Rightarrow \neg(a \mathcal{B} c)$,
—connected (strongly complete or total), if, $\forall a, b \in S$, $(a \mathcal{B} b) \vee (b \mathcal{B} a) \vee (a = b)$.

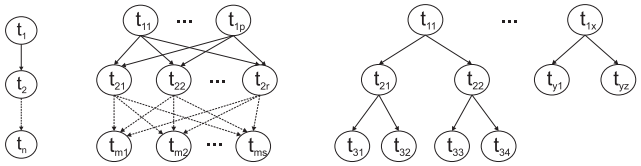
The above properties are not independent. For instance, asymmetry implies irreflexivity, while irreflexivity and transitivity imply asymmetry. In terms of a preference relation over a relational schema R, there is a subtle point regarding the set S over which the conditions of each property are tested. Typically, we should consider as S the set of all tuples $t = (u_1, u_2, \ldots, u_d)$, $u_i \in dom(A_i)$ of $R(A_1, A_2, \ldots A_d)$. However, in the presence of integrity constraints, we could apply the conditions only amongst tuples that all belong to a *valid instance* r of R, that is, to an instance r of R that does not violate any integrity constraints.

Based on its properties a preference relation $\succ_P$ is characterized as follows:

—A binary relation is a *preorder* or *quasi order*, if it is reflexive and transitive. If in addition, it is antisymmetric then it is a *partial order*.
—A binary relation is a *strict partial order* (or *irreflexive partial order*), if it is irreflexive, asymmetric and transitive. A preference relation $\succ_P$ over a relational schema R is usually a strict partial order.
—A binary relation is a *total order*, if it is a strict partial order and it is also connected. If a preference relation $\succ_P$ is a total order, any two tuples in any instance r of R are mutually comparable under $\succ_P$.
—A binary relation is a *weak order*, if it is a negatively transitive strict partial order.

A preference relation over an instance r of R can be represented through a directed graph that we call a *preference graph*. In the preference graph, there is one node for each tuple t in r and there is a directed edge from the node representing tuple t_i to the node representing tuple t_j, if and only if, $t_i \succ_P t_j$. Some properties of the preference relation have a counterpart graph property.

If the preference relation is transitive, it is common to represent the transitive reduction of the relation. In particular, there is an edge from t_i to t_j, if and only if, $t_i \succ_P t_j$ and $\nexists t_k$, such that, $t_i \succ_P t_k$ and $t_k \succ_P t_j$. The graph for a partially ordered set



(a) Total order (b) Weak order (c) Strict partial order

Fig. 3. Examples of preference graphs.

is also known as the *Hasse diagram*. In the following, we assume that preference relations are transitive and use the preference graph of their transitive reduction to represent them, unless stated otherwise. Examples of preference graphs for different types of preference relations are depicted in Figure 3.

Besides the explicit listing of preference relations between tuples, a convenient way to express preferences between tuples is by using logical formulas to express the constraints that two tuples must satisfy so that one is preferred over the other [Chomicki 2003].

DEFINITION 2. *Given a relational schema R and an instance r of R, a preference formula* $PF(t_i, t_j)$ *defines a preference relation* $\succ_P$ *over R, such that, for any pair of tuples* $t_i, t_j \in r$, $t_i \succ_P t_j$, *if only if,* $PF(t_i, t_j)$ *holds.*

Preference formulas allow us to express choices between sets of tuples in a single preference specification, and in this respect, they can be considered set-oriented.

Example 1: Consider the relation *movie* in Figure 1 and its instance shown in Figure 2. A user, say Addison, prefers a movie t over a movie t', if and only if, they are both of the same genre and t is longer than t' (preference P_1). P_1 can be expressed as follows. Given two tuples $t_i, t_j \in r$, $t_i \succ_{P_1} t_j$, if and only if, (t_i[*genre*] = t_j[*genre*] $\wedge$ t_i[*duration*] > t_j[*duration*]). Thus, for example, in the given instance, t_3 is preferred over t_1 under P_1.

Based on the form of the formula *PF*, Chomicki [2003] makes a distinction between intrinsic and extrinsic preferences. In *intrinsic preferences*, *PF* specifies conditions that refer solely to the values of the two database tuples t_i and t_j that are being compared. In *extrinsic preferences, PF* involves conditions that cannot be tested using only the values of the two tuples, such as the existence of other tuples in *r*, join conditions with tuples in other relations or comparisons of aggregate values. Using intrinsic preferences decouples the complexity of determining the preference order from the size of the database, since this involves only the two tuples being compared. It also makes the order insensitive to database updates.

Typically, intrinsic preference formulas are restricted to first order quantifier-free formulas expressed in a disjunction of conjunctions normal form (DNF) of simple equality or rational-order constraints among the values of the tuples. Specifically, a preference formula *PF* is a DNF of predicate conditions, $PF = (Cond_{11} \wedge \ldots \wedge Cond_{1m}) \vee \ldots \vee (Cond_{p1} \wedge \ldots \wedge Cond_{pb})$, where each predicate $Cond_{xy}$ has the form: (*i*) $t_i[A_v]\ \theta_v\ t_j[A_v]$ or (*ii*) $t_k[A_v]\ \theta_v\ co$, where $t_i, t_j \in r$, $k \in \{i, j\}$, $A_v \in A$, $co \in dom(A_v)$ and $\theta_v \in \{=, \neq, <, >, \leq, \geq\}$ if A_v is a numerical attribute and $\theta_v \in \{=, \neq\}$ otherwise.

Example 2: Addison prefers a recently produced drama movie or a horror movie with long duration (preference P_2). P_2 can be expressed as follows. Given two tuples $t_i, t_j \in r$, $t_i \succ_{P_2} t_j$, if and only if, (t_i[*genre*] = t_j[*genre*] $\wedge$ t_i[*genre*] = '*drama*'

$\wedge\ t_i[year] > t_j[year]) \vee (t_i[genre] = t_j[genre] \wedge t_i[genre] =$ '*horror*' $\wedge\ t_i[duration] > t_j[duration]$). In the movie instance of Figure 2, t_3 is preferred over t_1 under P_2.

Chomicki [2003] provides a characterization of the complexity of checking whether a preference relation defined using logical formulas satisfies specific properties of binary relations.

A formal language for formulating preference relations which are strict partial orders is proposed by Kießling [2002]. The language offers a number of base preference constructors, such as *POS* and *NEG*. For instance, *P* is a *POS*(*B, POS-set*) preference if: $\forall\ t_i, t_j \in r$, $t_i \succ_P t_j$, if and only if, $t_i[B] \in$ *POS-set* $\wedge\ t_j[B] \notin$ *POS-set*, where *POS-set* $\subseteq$ *dom*(*B*). This means that a desired value of *B* is one that belongs to a finite set *POS-set* of favorite values. For example, given the movie instance of Figure 2 and preference *POS*({*director*}, {*M. Curtiz*}), the most preferred tuple is the movie *Casablanca*. Similarly, a *NEG* preference defines that a desired value is one that does not belong to a finite set *NEG-set* $\subseteq$ *dom*(*B*) of dislikes. Preference constructors can be expressed using logical formulas.

2.1.2 *Quantitative Preferences.* In the quantitative approach, preferences are specified using functions that associate a numerical score with each database tuple (e.g., [Agrawal and Wimmers 2000]). The score expresses the importance or degree of interest in the tuple.

Definition 3. *Given an attribute set A, a preference function is a function* $f_P : dom(A) \rightarrow \mathbb{R}$ *that maps each tuple* $t \in dom(A)$ *to a real number called score.*

In general, t_i is preferred over t_j, that is, $t_i \succ_P t_j$ for a preference function f_P, if and only if, $f_P(t_i) > f_P(t_j)$.

Example 3: Given a scoring function $f_P(t_i) = 0.001 \times t_i[duration]$, tuples t_1, t_2 and t_3 in Figure 2 have interest scores 0.102, 0.109 and 0.195, respectively. Hence, t_3 is preferred over t_2, which in turn is preferred over t_1.

A convenient way to describe preferences is by specifying constraints that tuples must satisfy and assigning a preference score to these constraints. Preferences can be expressed as scores (or *degrees of interest*) to selection conditions (e.g., [Koutrika and Ioannidis 2004]). Preference scores belong to a predefined numerical domain which typically corresponds to the real interval [0, 1].

Definition 4. *Let* $R(A_1, \ldots, A_d)$ *be a relational schema and* $dom(A_i)$ *be the domain of attribute* A_i, $1 \leq i \leq d$. *A preference P on R is a pair* ($Condition_P$, $Score_P$), *where*

(i) $Condition_P$ *is of the form* $A_{n_1}\theta_{n_1}a_{n_1} \wedge \ldots \wedge A_{n_k}\theta_{n_k}a_{n_k}$ *and specifies a conjunction of simple selection conditions on the values* $a_{n_m} \in dom(A_{n_m})$ *of attributes* A_{n_m}, $n_m \in \{1, d\}$, *where* $\theta_{n_m} \in \{=, <, >, \leq, \geq, \neq\}$ *for numerical and* $\theta_{n_m} \in \{=, \neq\}$ *for categorical attributes.*

(ii) $Score_P$ *belongs to a predefined numerical domain.*

The meaning of such a preference is that all tuples from *R* that satisfy $Condition_P$ are assigned the interest $Score_P$. Clearly, a tuple *t* may satisfy the $Condition_P$ part of more than one preference *P*, thus more than one score may be assigned to it. We discuss ways to combine such scores towards assigning a single score to tuples later in Section 3. Let us denote the (combined) score assigned to a tuple *t* by *score*(*t*). A tuple t_i is preferred over a tuple t_j, that is, $t_i \succ t_j$, if and only if, $score(t_i) > score(t_j)$.

Example 4: Preference (*movie.genre* = '*drama*', 0.9) shows high interest in dramas, while preference (*movie.year* > 1990, 0.8) shows interest in recent movies.

Analogously to qualitative preferences, quantitative preferences can also be distinguished as intrinsic and extrinsic. Definition 4 describes intrinsic quantitative preferences, i.e., a preference score is assigned to a set of tuples based solely on the values of the tuple. Koutrika and Ioannidis [2004] support extrinsic preferences by allowing preferences for tuples in a relation R to be formulated based on values of attributes in different relations that join to R. To this end, Definition 4 is generalized as follows: a preference P on a relation R is a pair ($Condition_P$, $Score_P$), where $Condition_P$ is a conjunction of atomic selections involving a set of attributes B in the database and atomic joins transitively connecting these attributes to R.

Example 5: Consider the database schema in Figure 1. A preference for movies can be specified using attributes of movies, such as the year of a movie, but also attributes that are implicitly associated with movies and are stored in other relations, such as the names of the actors starring in a movie. For instance, a preference for movies with J. Roberts could be stated as follows:
(*movie.mid* = *play.mid* and *play.aid* = *actor.aid* and *actor.name* = '*J. Roberts*', 0.8).

2.1.3 *Equally Preferable and Incomparable Tuples.* Ideally, preferences should exist for all pairs of objects in a domain of interest. However, user preferences are typically incomplete. In such cases, the lack of a preference relation between two tuples may be either interpreted as an equal preference or attributed to incomparability or incompleteness. Equal preference means that the two tuples are equally preferred and in this sense equivalent to each other. Incomparability represents that two tuples cannot in some fundamental sense be compared with each other and it is likely to arise when tuples combine together multiple features or when tuples are essentially very different (for example, a car and a boat). Incompleteness, on the other hand, represents a gap in our knowledge of user preferences. In general, it is not always possible to differentiate among these three cases.

Any preference relation $\succ_P$ over a relational schema R induces an *indifference relation* $\sim_P$, such that, $\forall\ t_i, t_j \in r$, $t_i \sim_P t_j$, if and only if, $(\neg(t_i \succ_P t_j) \wedge \neg(t_j \succ_P t_i))$. Note that for a quantitative preference with scoring function f_P, $t_i \sim_f t_j \Leftrightarrow f_P(t_i) = f_P(t_j)$. Let us first look into specific types of a preference relation. If $\succ_P$ is a total order, then the indifference relation $\sim_P$ reduces to equality: $\forall\ t_i, t_j \in r$, $t_i \sim_P t_j \Leftrightarrow t_i = t_j$. In this case, there is no gap in our knowledge. If $\succ_P$ is a weak order, then indifference is an equivalence relation. A binary relation is an *equivalence relation* if it is reflexive, symmetric and transitive. Let $r/\sim_P$ be the set of equivalence classes defined over each instance r by $\sim_P$. This set is totally ordered by $\succ^*$, where $\succ^*$ is defined as: $\forall\ r_1, r_2 \in r/\sim_P$, $r_1 \subseteq r$, $r_2 \subseteq r$, $r_1 \succ^* r_2$, if and only if, $t_i \succ_P t_j$ for some $t_i \in r_1$ and $t_j \in r_2$. An example is shown in Figure 4. In this case, all indifferent tuples in a class are in a sense equivalent or equally preferred to each other.

However, if the preference relation $\succ_P$ is neither a total nor a weak order, the indifference relation $\sim_P$ may not be transitive, thus it may not be an equivalence relation. To see this, consider the graph shown in Figure 3c. Tuple t_{22} is indifferent to t_{21}, t_{21} is indifferent to t_{33}, but t_{22} is preferred over t_{33}. In this case, the indifference relation induced by the preference relation fails to capture the distinction between two tuples being incomparable versus being equally preferred. For example, the preference "I like drama and horror movies the same" should be interpreted differently from the preference "I cannot compare drama movies to horror movies". Next, we discuss how to make this distinction explicit for general

types of a preference relation.

When $\succ_P$ is a strict partial order, Kießling [2005] partitions the indifference relation $\sim_P$ into two parts: (*i*) a substitutable part and (*ii*) an alternative part. The *substitutable* part $\simeq_P$ includes tuples that can substitute each other in any preference expression and intuitively corresponds to the tuples that are equally preferred. Formally, the *substitutable relation* $\simeq_P$ is defined as: $\forall\ t_i, t_j \in r$, $t_i \simeq_P t_j$, only if, (*i*) $t_i \simeq_P t_j \Rightarrow t_i \sim_P t_j$, (*ii*) $\forall\ t_k \in r$, $t_k \succ_P t_i \Rightarrow t_k \succ_P t_j$, (*iii*) $\forall\ t_k \in r$, $t_i \succ_P t_k \Rightarrow t_j \succ_P t_k$ and (*iv*) if we interchange t_i and t_j in (*ii*) and (*iii*), they still hold. The substitutable relation is reflexive, symmetric and transitive. Two indifferent tuples that are not substitutable are called *alternatives*.

Example 6: In Figure 3c, t_{31} can substitute t_{32}, and t_{33} can substitute t_{34}, whereas, for instance, t_{21} and t_{22} are alternative tuples. As another example, consider the preference graph of Figure 5, representing a strict partial order among six tuples. Here, tuples t_{11} and t_{12} are substitutable with each other and so are t_{21} with t_{22}, while for example, t_{11} and t_{21} are alternatives. Note that when $\succ_P$ is a weak order, all tuples indifferent to each other (i.e., all tuples in the same equivalence class) are substitutable. For instance, tuples $t_{21}, t_{22}, \ldots, t_{2r}$, in Figure 3b, are substitutable.

To differentiate between equally preferable and incomparable tuples, we could define a *strong indifference relation* [Fishburn 1999], such that, $t_i \approx_P t_j$, if and only if, $\forall\ t_k \in r$, $t_i \sim_P t_k \Leftrightarrow t_j \sim_P t_k$. For example, t_{11}, t_{12} and t_{21}, t_{22}, in Figure 5, are strongly indifferent, whereas t_{11}, t_{21} are not. Tuples related through strong indifference correspond to substitutable ones, if transitivity holds. Strong indifference $\approx_P$ is an equivalence relation for any type of preference relation $\succ_P$. Let $r/\approx_P$ be the set of equivalence classes defined over each instance r by $\approx_P$. In this case, order $\succ^*$ defined over this set, $r/\approx_P$, of equivalence classes is not a total order.

An alternative way to differentiate between equally preferable and incomparable tuples is, instead of defining a preference order $\succ_P$, to define another binary relation $\succsim_P$ over R, such that, for two tuples t_i and t_j, $t_i \succsim t_j$ means that t_i is at least as good as or *at least as preferable* as t_j (e.g., [Georgiadis et al. 2008]). The "at least as preferable" relation $\succsim$ is normally a preorder (i.e., reflexive and transitive). The binary relation $\succsim_P$ induces a preference relation $\succ_P$ over R, such that, $\forall\ t_i, t_j \in r$, $t_i \succ_P t_j$, if and only if, $(t_i \succsim_P t_j \wedge \neg(t_j \succsim_P t_i))$. Note that this corresponds to the asymmetric part of $\succsim$. From the symmetric part of $\succsim$, we get an incomparability relation $\|_P$ as follows: $\forall\ t_i, t_j \in r$, $t_i \|_P t_j$, if and only if, $(\neg(t_i \succsim_P t_j) \wedge \neg(t_j \succsim_P t_i))$. We also get an equally preferable relation $\simeq_P$ as follows: $\forall\ t_i, t_j \in r$, $t_i \simeq_P t_j$, if and only if, $((t_i \succsim_P t_j) \wedge (t_j \succsim_P t_i))$. If $\succsim_P$ is a preorder, $\succ_P$ is a strict partial order. In addition, the equally preferable relation $\simeq_P$ is transitive and thus, forms an equivalence relation. If $\succsim_P$ is a preorder that is also antisymmetric, that is, if $\succsim_P$ is a partial order, then $\simeq_P$ is equality. Finally, if $\succsim_P$ is a preorder that is also connected, $\succ_P$ is a weak order and $\|_P$ is empty, that is, all tuples are comparable.

2.1.4 *Qualitative vs. Quantitative Preferences.* Qualitative preferences are described in a relative way through explicit tuple comparisons, while quantitative preferences are expressed in an absolute way directly on the desired tuples. In terms of expressive power, the qualitative specification of preferences is more general than the quantitative one, since not all preference relations can be expressed by scoring functions or through degrees of interest in conditions.

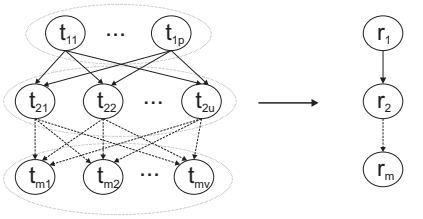


Fig. 4. Tuples t_{kl} are related through the weak order $\succ_P$. All tuples within a dotted oval are indifferent to each other and form an equivalence class r_k. The equivalence classes r_i are totally ordered by $\succ^*$.

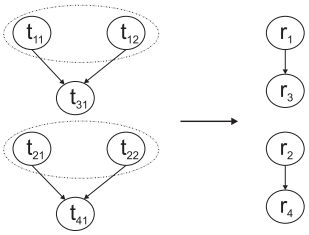


Fig. 5. Examples of substitutable and strongly indifferent tuples.

For example, take preference P_1 in Example 1: Addison prefers one movie tuple to another, if and only if, their genres are the same and the duration of the first is longer. Then in the example movie instance, t_3 is preferred over t_1 and t_2 cannot be compared with any of them. This preference cannot be expressed quantitatively, i.e., there is no scoring function that captures it. For the purpose of contradiction, assume that there is such a scoring function. Since there is no preference defined between any of the tuples t_1, t_3 and t_2 in Figure 2, the score of t_2 should be equal to the scores of t_1 and t_3. But this implies that the scores of t_1 and t_3 are the same, which is not possible since t_3 is preferred over t_1. Note that $\succ_{P_1}$ is not a weak order.

It has been shown that when the set over which the preference relation is defined is countable, a necessary and sufficient condition for a scoring function f_P, such that, $t_i \succ_P t_j \Leftrightarrow f_P(t_i) > f_P(t_j)$ to exist, is that $\succ_P$ is a weak order [Fishburn 1999]. It is easy to see that if $\succ_P$ can be captured by a scoring function f_P, then it is a weak order. To prove sufficiency, assume that $\succ_P$ is a weak order. Then, the induced indifference relation $\sim_P$ is an equivalence relation. The trick for defining f_P is to assign scores to each tuple so that (*i*) tuples that belong to the same equivalence class get the same score and (*ii*) for two tuples t_i and t_j that belong to two different equivalence classes r_1 and r_2 respectively, $f_P(t_i) > f_P(t_j)$, if and only if, $r_1 \succ^* r_2$.

Since using f_P implies that the produced preference relation is a weak order, we cannot find for preference relations that are not weak orders, a function f_P, such that, $t_i \succ_P t_j \Leftrightarrow f_P(t_i) > f_P(t_j)$. A less strict condition exists to preserve $\succ_P$ one way that is: $t_i \succ_P t_j \Rightarrow f_P(t_i) > f_P(t_j)$. For the example preference P_1, we could for instance define f_P as, $f_P(t_1) = 1$, $f_P(t_2) = 2$ and $f_P(t_3) = 3$.

In particular, it has been shown that, when the set over which the preference relation is defined is countable, a necessary and sufficient condition for a scoring function f_P, such that, $t_i \succ_P t_j \Rightarrow f_P(t_i) > f_P(t_j)$ and $t_i \approx_P t_j \Leftrightarrow f_P(t_i) = f_P(t_j)$ to exist, is that $\succ_P$ is acyclic [Fishburn 1999]. Relation $\succ_P$ is *acyclic* if we never have $t_1 \succ_P t_2 \succ_P \ldots \succ_P t_m \succ_P t_1$, for finite m. The idea here is to use the equivalence classes induced by strong indifference for assigning scores. In this case, the order $\succ^*$ among the equivalence classes is not a total order.

However, qualitative preferences return the most preferred tuples without distinguishing how much better one tuple is compared to another. For example, they cannot distinguish preferences, such as “I like comedies very much” vs. “I like dramas a little”. This also holds for negative preferences, i.e., there is no way to capture “I do not like adventures” vs. “I extremely dislike horrors”. Finally, it may

be easier for users to express relative pair-wise preferences between tuples than assigning scores. However, employing such preferences to attain an aggregate order of all involved tuples is clearly more complex than evaluating a scoring function.

2.2 Preference Granularity

Irrespectively of their (qualitative or quantitative) formulation, preferences may be expressed at different levels of granularity.

Tuple preferences are expressed between individual database tuples. Typically, they are formulated over the tuples of one or more relations based on the values of their attributes. In the previous subsection, we examined several such examples.

Set preferences depend not only on values of the individual tuples, but also on properties of groups of tuples as a whole. For example, Addison may want to select a set of three movies and prefers one of them to be a comedy, or as many of them as possible to have the same director. This means that we need to specify preferences over all subsets of three movies. Zhang and Chomicki [2011] represent set preferences qualitatively through profiles. A profile of a subset of k tuples is defined as a tuple of features where each feature corresponds to a quantity of interest (i.e., the number of comedies or distinct directors in our example). Such features may be generated by an aggregate query such as *min*, *max*, *sum*, *count*, *avg* over subsets of the same cardinality k (i.e., of $k = 3$ in our example). Then, a set preference can be formulated as a tuple preference over profiles of sets with the same cardinality.

Attribute preferences express preferences between attributes. They can have different interpretations. For example, they can be used to set priorities among tuple preferences based on the attributes involved in the preferences (e.g., [Georgiadis et al. 2008]).

Example 7: Addison prefers director S. Spielberg over A. Hitchcock and horror over drama movies. She considers the director of a movie to be more important than its genre which is expressed through an attribute preference $director \succ genre$. For example, this results in tuple t_3 being preferred over t_2 in Figure 2.

Alternatively, attribute preferences can express priorities among the attributes to be displayed in the result of a query, such as in the quantitative attribute preferences, called π-*preferences*, discussed in the framework of [Miele et al. 2009]. A π-*preference* is expressed by assigning an interest score to an attribute. Das et al. [2006b] and Miah et al. [2008] propose automatic techniques for scoring attributes in search results based on the attribute usefulness and visibility.

Relationship preferences are expressed on relationships between two types of entities (called *generic*) or two particular entities (called *instance*). A generic relationship preference shows interest in a particular type of relationship. For example, one may consider significant the relationship "a director has directed movies". On the other hand, one may consider significant the relationship "J. Roberts acted in Ocean's Twelve". This is an example of an instance relationship preference.

A framework that supports both tuple and relationship preferences for different relations of a database schema is presented by Koutrika and Ioannidis [2004]. User preferences are stored as degrees of interest in atomic query elements that can be individual selection or join conditions (called *selection* and *join preferences*, respectively). Join preferences show user interest in tuples from a relation R that join to tuples in different relations for which preferences do exist.

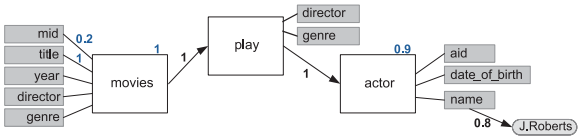


Fig. 6. Personalization graph.

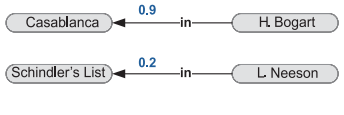


Fig. 7. Example instance relationship preferences.

Example 8: Addison's liking of the actress J. Roberts is expressed as a selection preference: (*actor.name* = '*J. Roberts*', 0.8). She also considers the actor of a movie very important and hence she has the following join preferences over the database schema of Figure 1: (*movie.mid* = *play.mid*, 1) and (*play.aid* = *actor.aid*, 1). These join preferences essentially connect movies to their actors and show that movie preferences are also shaped by preferences for their actors.

To map simple tuple and relationship preferences over a database, a *personalization graph*, $G(V,E)$, is introduced. $G(V,E)$ is a directed graph that is an extension of the database schema graph. Graph nodes correspond to schema relations, attributes and values for which a user has a preference. Edges are selection edges, representing a possible selection condition from an attribute to a value node, and join edges, representing a join between relations. Tuple preferences map to selection edges. Join edges capture generic relationship preferences. Figure 6 illustrates a personalization graph that captures the preferences given in Example 8. Similarly, one could imagine an instance-based personalization graph (e.g., the one in Figure 7), where nodes correspond to tuples and edges correspond to relationships between tuples. A personalization graph could also capture relation and attribute preferences expressed as degrees of interest (for example, as shown in different color in Figure 6).

Relation preferences are expressed on a class of entities. For example, "I am interested in directors, but not in producers". To the best of our knowledge, relation preferences are not found in the literature so far.

Recently, Golfarelli et al. [2011] introduced preferences in the context of OLAP. Preferences are defined at the schema level, thus allowing preferences at the aggregation level of facts, i.e., on their group-by sets. The basic idea is to define preferences on the space of hierarchy attributes. These are used to induce preferences on the space of the corresponding facts, for instance, by stating that monthly data are preferred over yearly or daily data.

Observe that the granularity at which preferences can be expressed is determined to a great extent by the database schema. For example, consider that instead of the movie relation depicted in Figure 1, we have the following relations: *movie*(*mid, title, year, language, duration, did*), *director*(*did, name*), *genre*(*mid, genre*). Then, the preference for the *director* over the *genre* expressed as an attribute preference in Example 7, would be instead formulated as a relationship preference.

2.3 Context

In the preference formulation paradigms studied so far, preferences hold unconditionally, that is under all circumstances. Such preferences are called *context-free*. *Contextual* or *conditional* preferences hold under specific circumstances and, in their general form, consist of two parts, namely a context and a preference one:

DEFINITION 5. *A contextual preference CP is a pair (C, P), where C defines the context and P defines the preference.*

The context part C specifies the conditions under which preference P holds, where P may be specified either qualitatively or quantitatively. In this section, we focus on the context part.

Adapting the behavior of an application to take the current user context into account has been studied extensively in the context-aware computing community, where different definitions of context have been proposed (e.g., [Brown et al. 1997; Schmidt et al. 1999]). A generic definition is the following [Dey 2001]:

"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place or object that is considered relevant to the interaction between a user and an application, including the user and application."

Under this general definition, user preferences can be also considered part of the user context because they characterize the situation of a user and can be used to modify queries. Here, we study how context determines when user preferences hold. With these observations in mind, we present our own definition of context:

DEFINITION 6. *Context is any external to the database information that can be used to characterize the situation of a user or any internally stored information that can be used to characterize the data per se.*

Based on this definition, context can be dictated by our data (i.e., internal context) or it can be ephemeral, volatile and external to the database. On the basis of this distinction, contextual preferences fall into two categories: *internal contextual preferences* and *external contextual preferences*. For example, an internal contextual preference can be that, for animations, the year of release should be after 2000. On the other hand, a preference for thrillers after midnight is an external contextual preference and, in particular, a time-dependent one.

2.3.1 *Internal Contextual Preferences.* A simple way to specify the context part C of an internal contextual preference (C, P) is by specifying conditions for the presence of specific attribute values ([Agrawal et al. 2006; Chomicki 2003]). Formally:

DEFINITION 7. *Given a relation schema $R(A_1, A_2, \ldots, A_d)$, an internal context C is a formula of the form $\wedge_{j \in L}(A_j \; \theta \; a_j)$ where $L \subseteq \{1, \ldots, d\}$ and $a_j \in dom(A_j)$.*

For an instance r of R, the meaning of a preference (C, P) where C is defined as above is that preference P holds for all tuples $t \in r$, such that, $t[A_j] \; \theta \; a_{i_j}, \forall j \in L$.

Example 9: Consider that Addison prefers A. Hitchcock over M. Curtiz as a director of horror movies and M. Curtiz over A. Hitchcock as a director of drama movies. This can be be expressed with the following two contextual preferences, where preferences are specified qualitatively: CP_x = (genre = '*horror*', $t_i \succ_{P_x} t_j$, if and only if, t_i[*director*] = '*A. Hitchcock*' $\wedge$ t_j[*director*] = '*M. Curtiz*') and CP_y = (genre = '*drama*', $t_i \succ_{P_y} t_j$, if and only if, t_i[*director*] = '*M. Curtiz*' $\wedge$ t_j[*director*] = '*A. Hitchcock*').

Note that the above specification just offers notational convenience, since such preferences can also be expressed by the following qualitative preference: Given two tuples $t_i, t_j \in r$, $t_i \succ_P t_j$, if and only if, (t_i[*genre*] = t_j[*genre*] $\wedge$ t_i[*genre*] = '*horror*' $\wedge$ t_i[*director*] = '*A. Hitchcock*' $\wedge$ t_j[*director*] = '*M. Curtiz*') $\vee$ (t_i[*genre*] = t_j[*genre*] $\wedge$ t_i[*genre*] = '*drama*' $\wedge$ t_i[*director*] = '*M. Curtiz*' $\wedge$ t_j[*director*] = '*A. Hitchcock*').

While little work has been done in databases, conditional preferences have been studied extensively in AI. A commonly used graphical notation for their repre-

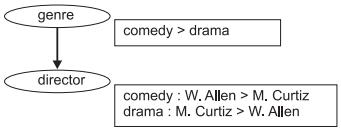


Fig. 8. CP-net example.

sentation is a *conditional preference network*, or *CP-net* (e.g., [Boutilier et al. 2004]). CP-nets use conditional *ceteris paribus* (all else being equal) semantics. A CP-net over a set of attributes $A = \{A_1, \ldots, A_d\}$ is a directed graph in which there is a node for each attribute in A. If an edge from an attribute A_j to an attribute A_i exists in the graph, then A_j is an ancestor of A_i. Let Z_i be the set of all ancestors of A_i. Semantically, the preferences over A_i depend on the attributes Z_i. Each attribute A_i is annotated with a *conditional preference table*, *CPT*, describing the preferences over A_i's values given a combination of its ancestor values. That is, the *CPT* of A_i contains a set of preference expressions of the form $z_i : a_{i_1} \succ a_{i_2}$, $z_i \in dom(Z_i)$ and $a_{i_1}, a_{i_2} \in dom(A_i)$. This statement defines the conditional preference of a_{i_1} over a_{i_2} under context z_i. In particular, a tuple with a_{i_1} is preferred over a tuple with a_{i_2} when the value of Z_i is z_i, only if the values of the remaining attributes are equal.

Example 10: Let us assume that Addison prefers comedies over dramas. Then, in case of comedies, she prefers comedies directed by W. Allen over those directed by M. Curtiz and in case of dramas, she prefers those directed by M. Curtiz over those directed by W. Allen. Figure 8 depicts the CP-net for these preferences.

There are also a few recent proposals of using CP-nets to represent database preferences. *Hierarchical CP-nets* introduced in [Mindolin and Chomicki 2007] extend CP-nets by adding, among others, attribute preferences; a preference over an attribute is of higher priority than the preferences over the descendants of this attribute. Thus, each edge of the net expresses both the conditional dependence and the relative importance between different attributes. Ciaccia [2007] considers *incomplete CP-nets* where preferences are only partially specified resulting in pairs of tuples being incomparable or indifferent in some contexts. For instance, assume that in the previous example, the preference for directors in the context of comedies is not specified. This approach proposes decoupling CP-nets from the ceteris paribus semantics used so far, and using the totalitarian semantics, that intuitively consider a tuple t_i to be preferred over t_j if the value of t_i at some attribute is preferred over the corresponding value of t_j and none of the other values of t_j is preferred over that of t_i (Pareto semantics). Finally, Endres and Kießling [2006] consider translating a CP-net into an expression in the formal preference language over strict partial orders of Kießling [2002] and introduce a new preference constructor to capture the ceteris paribus semantics.

2.3.2 *External Contextual Preferences.* In an external contextual preference (C, P), the context part C describes a situation outside the database. Common types of external context include the *computing context* (e.g., network connectivity, nearby resources), the *user context* (e.g., profile, location), the *physical context* (e.g., noise levels, temperature) and *time* [Chen and Kotz 2000].

A simple way to model external context is through a finite set of special-purpose attributes, called context parameters:

DEFINITION 8. *Given a set of context parameters* $C_1, \ldots, C_n$ *with domains* $dom(C_1)$,

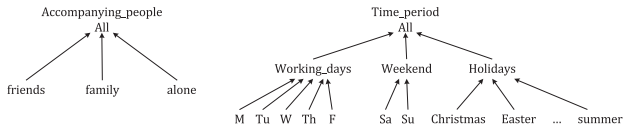


Fig. 9. Context hierarchies.

..., dom(C_n), respectively, an external context C is an n-tuple of the form ($c_1, \ldots, c_n$), where $c_i \in dom(C_i)$.

Stefanidis et al. [2006] propose using context parameters that take values from hierarchical domains thus allowing the definition of contextual preferences at various levels of detail, for example preferences that hold at the level of a day or a month. This model includes only a single context parameter in a context descriptor. Stefanidis et al. [2007a] and Miele et al. [2009] describe contextual preferences using more than one context parameter and over a set of external contexts.

Example 11: Assume the context parameters *accompanying_people* and *time_period* and their corresponding hierarchies depicted in Figure 9. Say Addison enjoys comedies when accompanied with friends during holidays. The context part of such a preference is defined by (*friends, holidays*).

External contexts, termed *situations*, are also discussed by Holland and Kießling [2004]. Each context has an identifier *cid* and consists of a timestamp and location as well as other influences, such as physical state, current emotion, weather conditions and accompanying people. External contexts are uniquely linked through an N:M relationship with preferences. Therefore, each external contextual preference can be described as a (*cid, pid*) relationship instance, expressing that preference *pid* holds in context *cid*.

Finally, Bunningen et al. [2006] propose a knowledge-based contextual preference model. Contextual preferences, called *preference rules*, have the form (C, P), where both context and preference are description logics concept expressions.

2.4 Preference Aspects

Preferences naturally come into different flavors and express different opinions and desires. For example, a preference may express like (e.g., "I like going to movies") or dislike (e.g., "I do not like long movies"). It may capture a general rule (e.g., "I like all comedies") or a finer-grained taste (e.g., "I like comedies released after 2000 by American directors") and so forth. In what follows, we present a set of dimensions and several types of preferences with the purpose of further understanding the expressivity of existing preference models.

Uncertainty expresses our level of confidence on whether a particular preference holds. To model uncertainty, fuzzy set theory [Zadeh et al. 1975; Zimmermann 1985] is often used to represent qualitative fuzzy preferences (e.g., [Orlovsky 1978; Ovchinnikov and Roubens 1992]). Given a relational schema $R(A_1, \ldots, A_d)$ with $A = \{A_1, \ldots, A_d\}$ and $dom(A) = dom(A_1) \times \ldots \times dom(A_d)$, a *fuzzy preference relation FP* is a pair ($dom(A) \times dom(A)$, μ_{FP}), where μ_{FP}: $dom(A) \times dom(A) \rightarrow [0, 1]$ is a membership function that maps each pair of tuples (t_i, t_j) to [0, 1]. Intuitively, $\mu_{FP}(t_i, t_j)$ indicates the credibility of the preference $t_i \succ_{FP} t_j$. For two tuples t_i, t_j, $\mu_{FP}(t_i, t_j) = 0$ means that $t_i \succ_{FP} t_j$ does not belong to the preference relation, whereas $\mu_{FP}(t_i, t_j) = 1$ indicates that $t_i \succ_{FP} t_j$ holds; all other values of μ_{FP} indicate fuzzy membership. Note that for non-fuzzy preferences as defined by Definition

1, $\mu_{FP}(t_i, t_j) \in \{0, 1\}$, for all pairs (t_i, t_j) in $dom(A) \times dom(A)$. The typical properties of binary relations are translated to the fuzzy case. For example, for reflexivity, $\mu_{FP}(t_i, t_i) = 1$, $\forall t_i \in dom(A)$, while for irreflexivity, $\mu_{FP}(t_i, t_i) = 0$, $\forall\ t_i \in dom(A)$.

Intensity shows the degree of desire expressed in a preference and answers the question of "how strong a preference is". Synonym concepts are priority, significance and importance. Preferences can be loosely characterized as *strong*, *weak* or *moderate*. As we have discussed in Section 2.1, quantitative approaches capture intensity in their score or degree of interest. For example, the preference ($movie.genre$ = '*drama*', 0.9) is a strong preference compared to the weaker ($movie.genre$ = '*animation*', 0.3). On the other hand, qualitative approaches can show intensity only in an abstract way by the virtue of comparison.

Necessity corresponds to the satisfaction of a preference and answers to the question "should the preference be met". Whereas, in the traditional Boolean database query model, query conditions are by default considered *hard* constraints, a preference may be seen as *optional*. The degree of interest can again be used as an indication of the necessity of a preference, however, an explicit distinction between hard and optional preferences has also been proposed in the preference language of [Kießling and Köstler 2002].

Feeling answers to the question "how one feels about something". Preferences may be *positive*, i.e., expressing like, *negative*, expressing dislike or *neutral* if they convey no particular taste. Neutrality can be explicitly captured by associating specific score values with it, for instance, in Koutrika and Ioannidis [2005b] degree of interest equal to 0 or in Agrawal and Wimmers [2000] the symbol $\perp$.

Complexity describes the degree of detail of a preference expression and answers to the question "how specific a preference is". A preference may be *generic* or *simple* if it refers to a single relationship or attribute of the entities of interest. For example, a qualitative preference that compares a pair of entities based on a single attribute is a simple preference. A *compound* preference jointly expresses a combination of simple preferences to be concurrently met. For example, a preference for "comedies directed by W. Allen after 2000" is a compound one expressing a very specific interest in a subclass of comedies [Chomicki 2003].

Attitude indicates whether a preference is associated with the existence (*presence* preference) or absence of certain values or relationships (*absence* preference). For instance, a preference for a movie being a comedy is a presence preference whereas a preference for movies without violence is an absence one. Also, a veto expresses a prohibition on the presence of a specific set of values in the elements of the answer to a query [Agrawal and Wimmers 2000; Chomicki 2003].

Elasticity indicates how strict a preference is. An *exact* preference is either satisfied exactly or not at all. *Elastic* preferences may be satisfied as closely as possible. Kießling [2002] captures elastic preferences using special order and distance operators. For example, the $AROUND(B, z)$ preference constructor denotes that tuples whose value in B has the smallest distance from z are preferred. Another example elastic preference is ($movie.duration = 120min, Score(movie.duration, 120)$), where $Score()$ is a function based on the distance of the actual duration of the movie from the desired duration [Koutrika and Ioannidis 2005b]. Elasticity can be represented also qualitatively. For example, ($\forall\ t_i,\ t_j \in r,\ t_i \succ_P t_j$, if and only if, $t_i[year] >$

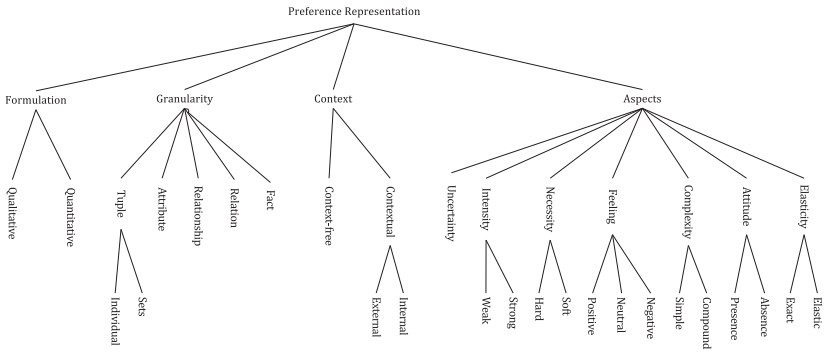


Fig. 10. A taxonomy of preference representations.

$t_j[year] \wedge t_j[year] > 2000$) expresses a preference on recent movies as long as they are more recent than year 2000.

Note also that approximate queries, such as fuzzy (e.g., [Fagin 1998]), or vague (e.g., [Motro 1988]) queries, can be seen as a way to specify elastic preferences by specifying target values. Again, in this case, tuples are ranked according to how close or similar they are to the targets, based on appropriately defined distance metrics over their data values.

2.5 Summary of Preference Representations

In short, we can categorize the various approaches to preference representation based on the following dimensions: how preferences are formulated (*formulation*), at what level they are expressed (*granularity*), under which conditions they hold (*context*) and what they express (*aspects*). Preferences can be specified in either a qualitative or quantitative way over tuples, relations, attributes, relationships or facts. Context-free preferences hold under all possible contexts, whereas contextual preferences hold in a specific context that can be internal or external to the database. Finally, preferences can be characterized based on various aspects, such as intensity, feeling and complexity. Figure 10 summarizes the various options in each dimension.

Table I compares preference representation approaches based on the formulation, granularity and context dimensions. Table II drills down to preference types and compares approaches based on what preference aspects are captured at each granularity. Most approaches focus on tuple preferences, whereas uncertainty is not captured. As it can be seen, there are many aspects of preferences still to be explored. In general, as discussed, the qualitative approach is more general than the quantitative approach. However, by using scores, quantitative approaches can express explicitly "how much" one tuple is preferred over another, that is, the intensity of a preference. Both Chomicki [2003] and Kießling [2002] support qualitative preferences. Chomicki [2003] offers a logical framework for formulating preferences, whereas Kießling [2002] takes an algebraic approach by defining preferences as strict partial orders through the application of a number of preference constructors. We have also marked the work in [Kießling 2002] as quantitative, since there is an explicit mention of scoring functions as basic constructs and a combining function for composing them.

Table I. Preference representation approaches w.r.t. preference formulation, granularity and context.

	Formulation		Granularity					Context		
	Qualitative	Quantitative	Tuple	Relation	Attribute	Relationship	Fact	Context-free	Internal	External
[Lacroix and Lavency 1987]	✓		✓					✓		
[Zhang and Chomicki 2011]	✓		sets					✓		
[Chomicki 2002; 2003]	✓		✓					✓	✓	
[Kießling 2002]	✓	✓	✓					✓		
[Agrawal and Wimmers 2000]		✓	✓					✓		
[Koutrika and Ioannidis 2004; 2005b]		✓	✓			✓		✓		
[Agrawal et al. 2006]	✓		✓						✓	
[Mindolin and Chomicki 2007]	✓		✓		✓				✓	
[Ciaccia 2007]	✓		✓						✓	
[Endres and Kießling 2006]	✓		✓						✓	
[Stefanidis et al. 2006; 2007a]		✓	✓					✓		✓
[Holland and Kießling 2004]	✓		✓							✓
[Bunningen et al. 2006]		✓	✓							✓
[Georgiadis et al. 2008]	✓		✓		✓			✓		
[Miele et al. 2009]		✓	✓		✓					✓
[Golfarelli et al. 2011]	✓						✓	✓		

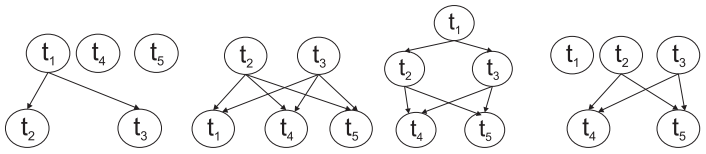
Table II. Preference representation approaches w.r.t. preference aspects (T=tuple, C=relation, A=attribute, R=relationship).

	Aspects												
	Intensity		Necessity		Feeling			Complexity		Attitude		Elasticity	
	strong	*weak*	*hard*	*soft*	*positive*	*negative*	*indifferent*	*simple*	*compound*	*presence*	*absence*	*exact*	*elastic*
[Lacroix and Lavency 1987]	T	T	–	T	T	–	–	T	T	T	–	T	–
[Chomicki 2002; 2003], [Zhang and Chomicki 2011]	T	T	–	T	T	–	T	T	T	T	T	T	T
[Kießling 2002]	T	T	–	T	T	T	–	T	T	T	T	T	T
[Agrawal and Wimmers 2000]	T	T	–	T	T	–	T	T	T	T	T	T	T
[Koutrika and Ioannidis 2004; 2005b; 2010]	T	T	TR	TR	T	T	T	TR	TR	T	T	T	T
[Agrawal et al. 2006]	T	T	–	T	T	–	–	T	T	T	–	T	–
[Stefanidis et al. 2006; 2007a]	T	T	–	T	T	–	–	T	T	T	T	T	–
[Holland and Kießling 2004]	T	T	–	T	T	T	–	T	T	T	T	T	T
[Bunningen et al. 2006]	T	T	–	T	T	–	–	T	T	T	T	T	–
[Georgiadis et al. 2008]	TA	TA	A	T	TA	–	TA	T	T	TA	–	TA	–
[Miele et al. 2009]	TA	TA	A	TA	TA	–	–	TA	TA	TA	T	TA	–

3. PREFERENCE COMPOSITION

Given a set of preferences, there is a number of different composition methods or mechanisms for combining them to determine a single preference among the affected tuples. This issue arises when aggregating different preferences of a single person (individual preference modeling) as well as when combining preferences of different people (group preference modeling). In the former case, the different preferences of a user may be based on a number of different criteria that need to be combined, whereas in the latter case the final ranking seeks to reach a consensus among the group members. Note that a set of preferences may order the same pair of tuples differently. This situation is sometimes called a *conflict*.

One way to distinguish composition methods is based on *attitude*. Let P_x and P_y be two preferences, we may have the following two attitudes. In the *overriding attitude*, one of the preferences, say P_x, is given priority over the other, meaning



(a) P_x (b) P_y (c) $P_x\&P_y$ (d) $P_x \otimes P_y$

Fig. 11. Example of prioritized and Pareto composition.

that P_y is applicable only when P_x is not. In the *combinatory attitude*, both P_x and P_y contribute to the final ranking. As with preference representation, composition mechanisms can also be distinguished as qualitative or quantitative. *Qualitative composition* mechanisms produce a final pair-wise order of the tuples, whereas *quantitative composition* mechanisms assign an aggregated score to each tuple.

In the rest of this section, we present qualitative (Section 3.1) and then quantitative (Section 3.2) mechanisms for composing tuple preferences. We also outline mechanisms from the related area of rank aggregation (Section 3.3). Finally, we discuss composing preferences of different granularity (Section 3.4).

3.1 Qualitative Composition

Next, we describe qualitative mechanisms for composing preferences defined qualitatively using preference relations. Such mechanisms are also applicable to preferences defined quantitatively using functions or degrees of interest. Simply, recall that each f_P defines a preference relation $\succ_P$, such that $f_P(t_i) > f_P(t_j) \Leftrightarrow t_i \succ_P t_j$.

3.1.1 *Prioritized Preference Composition.* Given two preference relations P_x and P_y, in prioritized composition, one of them, say P_x, is given priority over the other.

Definition 9. *Let P_x and P_y be two preference relations defined over the same relational schema R. The prioritized preference composition relation $\succ_{P_x\&P_y}$ is defined over R, such that, $\forall\ t_i, t_j$ of R, $t_i \succ_{P_x\&P_y} t_j$, if and only if, $(t_i \succ_{P_x} t_j) \vee (t_i \sim_{P_x} t_j \wedge t_i \succ_{P_y} t_j)$.*

The intuitive meaning of prioritized composition is: use P_y only if P_x is not applicable. An example is shown in Figure 11, where the preference graph in Figure 11c represents the preference relation resulting from the prioritized composition of P_x and P_y represented by the preference graphs shown in Figures 11a and 11b respectively. Consider also the following example that uses logical formulas.

Example 12: Addison prefers drama movies over horror movies (preference P_3) and long movies over short ones (preference P_4). P_3 can be defined using logical formulas as: $t_i \succ_{P_3} t_j$, if and only if, $t_i[genre] =$ '*drama*' $\wedge\ t_j[genre] =$ '*horror*'. P_4 can be expressed as: $t_i \succ_{P_4} t_j$, if and only if, $t_i[duration] > t_j[duration]$. The prioritized preference $P_3\&P_4$ can be defined as: $t_i \succ_{P_3\&P_4} t_j$, if and only if, $(t_i[genre] =$ '*drama*' $\wedge\ t_j[genre] =$ '*horror*'$) \vee (t_i[genre] \neq$ '*drama*' $\wedge\ t_i[duration] > t_j[duration]) \vee (t_j[genre] \neq$ '*horror*' $\wedge\ t_i[duration] > t_j[duration])$. Now, for the movie relation of Figure 2, under $P_3\&P_4$, t_3 is preferred over t_1 which is in turn preferred over t_2.

Prioritized composition may also be applied on preference relations defined over different relational schemas. In this case, it is called *lexicographical composition*.

Definition 10. *Let P_x and P_y be two preference relations defined over relational schemas R and R′ with attribute domains dom(A) and dom(A′) respectively. The lexicographical preference composition relation $\succ_{P_x\&P_y}$ defined over the Cartesian product $R \times$*

R' is a subset of dom$(A) \times$ dom(A'), such that $(t_i, t'_i) \succ_{P_x \& P_y} (t_j, t'_j)$, if and only if,

$$(t_i \succ_{P_x} t_j) \vee (t_i \sim_{P_x} t_j \wedge t'_i \succ_{P_y} t'_j),$$

where t_i, t_j are tuples of R and t'_i, t'_j tuples of R'.

A special form of prioritized composition is achieved through preference overriding. The general idea is that in cases where two preferences P_x and P_y are both applicable but one of them, say P_x is applicable to a subset of the cases that P_y is, then priority is given to P_x over P_y for this subset. To make this concrete, let P_x and P_y be two preferences on R defined using preference formulas PF_x and PF_y respectively (Definition 2), that order two tuples t_i and t_j differently, say $t_i \succ_{P_x} t_j$ and $t_j \succ_{P_y} t_i$. Preference P_x is called *specific*, if $\forall t_i, t_j$, $PF_x(t_i, t_j) \Rightarrow PF_y(t_j, t_i)$. Preference P_y is called *generic*. Preference P_x is given priority over P_y, that is, P_y is used only when P_x does not apply.

Example 13: Addison prefers comedies over westerns only when directed by W. Allen, i.e., ($\forall$ t_i, $t_j \in r$, $t_i \succ_{P_x} t_j$, if and only if, t_i[*genre*] = 'comedy' $\wedge$ t_i[*director*] = 'W. Allen' $\wedge$ t_j[*genre*] = 'western'), otherwise, she prefers westerns over comedies i.e., ($\forall$ t_i, $t_j \in r$, $t_i \succ_{P_y} t_j$, if and only if, t_i[*genre*] = 'western' $\wedge$ t_j[*genre*] = 'comedy'). The former is a specific preference applicable to a subclass of comedies overriding the latter generic preference about comedies, thus it is given priority.

When a preference P_x is defined using a condition C_{P_x} (Definition 4), we can view it as a possible conjunctive query Q_{P_x} which selects tuples from R that satisfy C_{P_x}. Given this correspondence, Koutrika and Ioannidis [2010] define preference overriding using conjunctive query subsumption, that is, query containment. Given two preferences P_x and P_y defined on the same relational schema R using condition C_{P_x} and C_{P_y} respectively, P_y is *overridden* by P_x, if Q_{P_x} is subsumed by Q_{P_y}, that is, for all database instances, every answer to Q_{P_x} is an answer to Q_{P_y}, i.e., $Q_{P_x} \subseteq Q_{P_y}$.

3.1.2 *Pareto Preference Composition.* In Pareto composition, the involved preference relations are considered equally important.

Definition 11. *Let P_x and P_y be two preference relations defined over the same relational schema R. The Pareto preference composition relation $\succ_{P_x \otimes P_y}$ is defined over R, such that, $\forall$ t_i, t_j of R, $t_i \succ_{P_x \otimes P_y} t_j$, if and only if,* $(t_i \succ_{P_x} t_j \wedge \neg(t_j \succ_{P_y} t_i)) \vee (t_i \succ_{P_y} t_j \wedge \neg(t_j \succ_{P_x} t_i))$.

Note that for two tuples t_1 and t_2 and a preference relation P, $\neg(t_1 \succ_P t_2) \equiv (t_2 \succ_P t_1 \vee t_1 \sim_P t_2)$. Intuitively, under Pareto composition, a tuple is better than (or dominates) another if it is at least as good (i.e., not worse) under one preference and strictly better under the other. For instance, given the preference graphs of two preferences P_x, P_y (Figures 11a, 11b), Figure 11d depicts the preference graph of $P_x \otimes P_y$. As another example, consider the following.

Example 14: The Pareto preference $P_3 \otimes P_4$ (P_3 and P_4 are defined above) can be defined as: $t_i \succ_{P_3 \otimes P_4} t_j$, if and only if, ($t_i$[*genre*] = '*drama*' $\wedge$ t_j[*genre*] = '*horror*' $\wedge$ t_i[*duration*] $\geq$ t_j[*duration*]) $\vee$ (t_i[*duration*] $>$ t_j[*duration*] $\wedge$ t_j[*genre*] $\neq$ '*drama*') $\vee$ (t_i[*duration*] $>$ t_j[*duration*] $\wedge$ t_j[*genre*] = '*drama*' $\wedge$ t_i[*genre*] $\neq$ '*horror*'). For the movie relation of Figure 2, under $P_3 \otimes P_4$, t_3 is preferred over t_1, and t_1, t_2 are incomparable.

Pareto composition is also applicable to relations defined over different schemas.

Definition 12. *Let P_x and P_y be two preference relations defined over the relational schemas R and R' with attribute domains dom(A) and dom(A'), respectively. The multidimensional Pareto preference relation $\succ_{P_x \otimes P_y}$ defined over the Cartesian product $R \times R'$ is a subset of dom$(A) \times$ dom(A'), such that $(t_i, t'_i) \succ_{P_x \otimes P_y} (t_j, t'_j)$, if and only if,*

$(t_i \succ_{P_x} t_j \wedge \neg(t'_j \succ_{P_y} t'_i)) \vee (t'_i \succ_{P_y} t'_j \wedge \neg(t_j \succ_{P_x} t_i))$, *where* t_i, t_j *are tuples of* R *and* t'_i, t'_j *tuples of* R'.

3.1.3 *Set-Oriented Preference Composition.* Set-oriented preference composition is applicable only between preferences defined over the same relational schema. The resulting relation corresponds to the intersection, set difference or union of the two preference relations. Recall that preference relations are defined as binary relations, i.e., as sets.

In particular, given the relations P_x and P_y, the intersection preference relation $\succ_{P_x \wedge P_y}$ corresponds to $\succ_{P_x} \cap \succ_{P_y}$. More precisely:

Definition 13. *Let* P_x *and* P_y *be two preference relations defined over the same relational schema* R. *The intersection preference relation* $\succ_{P_x \wedge P_y}$ *is defined over* R, *such that,* $\forall$ t_i, t_j *of* R, $t_i \succ_{P_x \wedge P_y} t_j$, *if and only if,* $t_i \succ_{P_x} t_j \wedge t_i \succ_{P_y} t_j$.

Example 15: The intersection preference $P_3 \wedge P_4$ can be expressed as: $t_i \succ_{P_3 \wedge P_4} t_j$, if and only if, $(t_i[genre] =$ '*drama*' $\wedge$ $t_j[genre] =$ '*horror*') $\wedge$ $(t_i[duration] > t_j[duration])$. For instance, for the movie relation in Figure 2, t_3 is preferred over t_2 under $P_3 \wedge P_4$.

The difference preference relation $\succ_{P_x - P_y}$ corresponds to set difference $\succ_{P_x} - \succ_{P_y}$, and the union preference relation $\succ_{P_x + P_y}$ corresponds to union $\succ_{P_x} \cup \succ_{P_y}$. Formally:

Definition 14. *Let* P_x *and* P_y *be two preference relations defined over the same relational schema* R. *The difference preference relation* $\succ_{P_x - P_y}$ *is defined over* R, *such that,* $\forall$ t_i, t_j *of* R, $t_i \succ_{P_x - P_y} t_j$, *if and only if,* $t_i \succ_{P_x} t_j \wedge \neg(t_i \succ_{P_y} t_j)$.

Definition 15. *Let* P_x *and* P_y *be two preference relations defined over the same relational schema* R. *The union preference relation* $\succ_{P_x + P_y}$ *is defined over* R, *such that,* $\forall$ t_i, t_j *of* R, $t_i \succ_{P_x + P_y} t_j$, *if and only if,* $t_i \succ_{P_x} t_j \vee t_i \succ_{P_y} t_j$.

Set-oriented composition can be extended to apply between preference relations defined over different but union-compatible relational schemas [Kießling 2002]. Let $\succ_{P_x}$ be a preference relation over R_x with $dom(A_x)$ and $\succ_{P_y}$ be a preference relation over R_y with $dom(A_y)$, where R_x and R_y are union-compatible and their domains are disjoint, e.g., $dom(A_x) \cap dom(A_y) = \emptyset$. The *linear sum* preference relation $\succ_{P_x \oplus P_y}$ is defined over a new relation schema R_w with domain $dom(A_x) \cup dom(A_y)$, such that, $\forall$ t_i, t_j in R_w, $t_i \succ_{P_x \oplus P_y} t_j$, if and only if, $t_i \succ_{P_x} t_j \vee t_i \succ_{P_y} t_j \vee (t_i \in dom(A_x) \wedge t_j \in dom(A_y))$.

3.1.4 *Transitive Closure.* The transitive closure of a preference relation is defined as follows [Chomicki 2003].

Definition 16. *Let* $\succ_{PR}$ *be a preference relation defined over a relational schema* R *and* t_i, t_j *be two tuples of* R. *The transitive closure of* $\succ_{PR}$ *is a preference relation* $\succ_{PR^*}$ *over* R *defined as:* $t_i \succ_{PR^*} t_j$, *if and only if,* $t_i \succ^n_{PR} t_j$, $n > 0$, *where*

(i) $t_i \succ^1_{PR} t_j \equiv t_i \succ_{PR} t_j$ *and*

(ii) $t_i \succ^{n+1}_{PR} t_j \equiv \exists t_k$, *such that* $t_i \succ_{PR} t_k \wedge t_k \succ^n_{PR} t_j$.

An important point is that, when preference relations are defined using formulas, the transitive closure is not defined as the closure of a finite relation as is the case of a database instance. Ross et al. [2007] introduce a constraint language for expressing the preference formulas PFs that allows comparison and a limited form of arithmetic. They prove that the transitive closure computation of a partial order preference relation expressed using their language terminates. They also provide estimations for the size of the transitive closure for Pareto, prioritized and intersection composition.

3.1.5 *Discussion.* Let us first see which of the various kinds of orders are preserved by each composition type [Chomicki 2003]. Prioritized and lexicographical composition preserve total and weak orders, but not strict partial orders. Pareto composition does not preserve weak, total or strict partial orders. Set-oriented composition operators do not preserve weak and total orders. Strict partial order is preserved by intersection but not by set difference or union, whereas, linear sum composition as well as union for disjoint domains (called disjoint union) preserve strict partial orders.

Note that the provided definitions of the various types of qualitative compositions use the indifference relation $\sim_P$. There are corresponding definitions for the substitutable relation $\simeq_P$ [Kießling 2005] and for the "at least as preferable" relation $\gtrsim_P$ [Georgiadis et al. 2008] (as defined in Section 2.1.3). For these definitions, strict partial orders are preserved. For example, lexicographical composition is extended for the "at least as preferable" relation as follows: [Georgiadis et al. 2008]: (*i*) $(t_i, t'_i) \succ_{P_x \& P_y} (t_j, t'_j)$, if and only if, $(t_i \succ_{P_x} t_j) \vee (t_i \simeq_{P_x} t_j \wedge t'_i \succ_{P_y} t'_j)$, (*ii*) $(t_i, t'_i) \simeq_{P_x \& P_y} (t_j, t'_j)$, if and only if, $t_i \simeq_{P_x} t_j \wedge t'_i \simeq_{P_y} t'_j$ and (*iii*) $(t_i, t'_i) \|_{P_x \& P_y} (t_j, t'_j)$ otherwise. Similarly, Pareto composition for the "at least as preferable" relation $\gtrsim_P$ is defined as [Georgiadis et al. 2008]: (*i*) $(t_i, t'_i) \succ_{P_x \otimes P_y} (t_j, t'_j)$, if and only if, $(t_i \succ_{P_x} t_j \wedge t'_i \gtrsim_{P_y} t'_j) \vee (t'_i \succ_{P_y} t'_j \wedge t_i \gtrsim_{P_x} t_j)$, (*ii*) $(t_i, t'_i) \simeq_{P_x \otimes P_y} (t_j, t'_j)$, if and only if, $(t_i \simeq_{P_x} t_j \wedge t'_i \simeq_{P_y} t'_j)$ and (*iii*) $(t_i, t'_i) \|_{P_x \otimes P_y} (t_j, t'_j)$ otherwise.

So far, we have defined qualitative composition between two preference relations. Since the result of composition is a new preference relation, clearly we can compose any number of preference relations by gradually applying a sequence of the same or different composition operators. However, the semantics of an n-ary composition, $n > 2$, depend on whether the corresponding operators are associative. For example, whereas prioritized composition is associative, Pareto composition is not. The above extension of Pareto composition is also associative.

In terms of treating conflicts, the different composition mechanisms reflect different user attitudes towards resolving them. For instance, union ignores conflicts and thus such conflicts need to be prevented if we want to obtain a preference relation which is a strict partial order. Prioritized composition resolves preference conflicts by consistently giving priority to one of the preference relations, whereas Pareto composition reconciles direct conflicts between pairs of tuples.

Chomicki [2007a] defines different types of conflicts based on whether when eliminated by prioritized or Pareto composition, they reappear if the resulting relation is closed by transitivity. A *0-conflict* between two preference relations $\succ_{P_x}$ and $\succ_{P_y}$ is a pair t_i, t_j, such that $t_i \succ_{P_x} t_j$ and $t_j \succ_{P_y} t_i$. A *1-conflict* between $\succ_{P_x}$ and $\succ_{P_y}$ is a pair t_i, t_j, such that $t_i \succ_{P_x} t_j$, and there exists $s_1, \ldots, s_k$, $k \geq 1$, such that $(t_j \succ_{P_y} s_1) \wedge \ldots \wedge (s_k \succ_{P_y} t_i)$ and $\neg(t_i \succ_{P_x} s_k) \wedge \ldots \wedge \neg(s_1 \succ_{P_x} t_j)$. A *2-conflict* between $\succ_{P_x}$ and $\succ_{P_y}$ is a pair t_i, t_j, such that there exists $s_1, \ldots, s_k$, $k \geq 1$, and $w_1, \ldots, w_m$, $m \geq 1$, such that $(t_j \succ_{P_y} s_1) \wedge \ldots \wedge (s_k \succ_{P_y} t_i)$, $\neg(t_i \succ_{P_x} s_k) \wedge \ldots \wedge \neg(s_1 \succ_{P_x} t_j)$, $(t_i \succ_{P_x} w_1) \wedge \ldots \wedge (w_m \succ_{P_x} t_j)$ and $\neg(t_j \succ_{P_y} w_m) \wedge \ldots \wedge \neg(w_1 \succ_{P_y} t_i)$. For strict partial orders, it was shown that prioritized composition resolves those 0-conflicts that are not 1-conflicts, while Pareto composition resolves those 1-conflicts that are not 2-conflicts.

3.2 Quantitative Composition

A general quantitative way of composing preferences that are expressed quantitatively is by applying a function to combine the scores assigned to a tuple by each of the preferences.

DEFINITION 17. *Given two preferences P_x, P_y over R defined through preference functions f_{P_x}, f_{P_y} respectively and a combining function $F : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$, $\forall\ t_i$, t_j in R, $t_i \succ_{rank_F(P_x,P_y)} t_j$, if and only if, $F(f_{P_x}(t_i), f_{P_y}(t_i)) > F(f_{P_x}(t_j), f_{P_y}(t_j))$.*

Example 16: Assume preference P_5 with scoring function $f_{P_5}(t_i) = 0.001 \times t_i[duration]$ and preference P_6 with scoring function $f_{P_6}(t_i) = 0.0001 \times t_i[year]$. A commonly used combining function is weighted summation. For example, the numerical preference $rank_F(P_5, P_6)$ with combining function $F(f_{P_5}(t_i), f_{P_6}(t_i)) = 0.1 \times f_{P_5}(t_i) + 0.9 \times f_{P_6}(t_i)$ combines the two preferences by assigning weight 0.1 to P_5 and weight 0.9 to P_6. In this case, tuples t_1, t_2, t_3 of the movie relation of Figure 2 get the combined scores 0.185, 0.187 and 0.199, respectively.

Besides weighted summation or average, other commonly used combining functions are "*min*" and "*max*". Agrawal and Wimmers [2000] propose a generic combine operator as part of their formal preference framework. The operator can be instantiated by different combining functions, called *value* functions, to yield specific instances of the combine operator. The result of combining preferences is a preference, thus providing the closure property.

In [Fagin 1996; 1998], several combining functions are considered for multimedia databases where each tuple is assigned a score $s \in [0, 1]$ based on how well it satisfies an atomic query. The objective is to compute an aggregated score for each tuple based on how well the tuple satisfies a boolean combination (disjunction or conjunction) of such atomic queries. A combining function F is a *triangular norm* if it is monotonous, commutative, associative and also preserves conjunction expressed by the following boundary conditions: $F(0, 0) = 0$ and $F(s, 1) = F(1, s) = s$. A combining function F is a *triangular co-norm*, if it is monotonous, commutative, associative and also preserves disjunction expressed by $F(1, 1) = 1$ and $F(s, 0) = F(0, s) = s$. Triangular norms and triangular co-norms are duals. For example the "*min*" function is a triangular norm with co-norm the "*max*" function. Another related property is *strictness*, where a combining function is strict if it takes its maximum value precisely when all its arguments take their maximum values.

Koutrika and Ioannidis [2005b] distinguish combining functions to be: (*i*) *inflationary*, when the score of a tuple that satisfies multiple preferences together increases with the number of these preferences; (*ii*) *dominant*, when the score assigned by one of the preferences dominates; and (*iii*) *reserved*, when the combined score of each tuple lies between the highest and the lowest scores assigned to it.

Independently of the type of the combining function, Fagin and Wimmers [2000] provide a formula for mapping an unweighted combination of preference functions to a weighted one. This formula has three properties: (*i*) if all weights are equal, the weighted combining function coincides with the unweighed one, (*ii*) if a weight is zero, the value of the function associated with this weight does not affect the resulting score and (*iii*) the resulting score is a continuous function of the weights.

Analogously to prioritized and Pareto composition, numerical composition may be applied to preferences defined over different schemas (e.g., [Hristidis and Papakonstantinou 2004; Ilyas et al. 2004]).

DEFINITION 18. *Given two preferences P_x, P_y defined over R, R' through preference functions f_{P_x}, f_{P_y}, respectively, and a combining function $F : \mathbb{R}\times\mathbb{R} \rightarrow \mathbb{R}$, $(t_i, t'_i) \succ_{rank_F(P_x,P_y)} (t_j, t'_j)$, if and only if, $F(f_{P_x}(t_i), f_{P_y}(t'_i)) > F(f_{P_x}(t_j), f_{P_y}(t'_j))$, where t_i, t_j are tuples of R and t'_i, t'_j are tuples of R'.*

3.3 Rank Aggregation

A topic related to preference composition is *rank aggregation*, which refers to the following problem: given *m* different ordered lists or rankings of a set of items, how to produce a single ranking of the items. An instance of rank aggregation known as *social choice* studies the problem of determining the ranking of alternatives that is best for a group given the individual opinions of its members. Social choice has been studied extensively in economics, politics, sociology, and mathematics (e.g., [Condorcet 1785; Taylor 1995]). Rank aggregation is also studied in the web, for example, in meta-search, where ranked lists of web pages produced by different search engines need to be combined into a single one (e.g., [Dwork et al. 2001; Cohen et al. 1999]. In database middleware, a related problem refers to finding the most preferred objects when there are multiple rankings based on preferences defined on different attributes or dimensions of these objects (e.g., [Fagin et al. 2001]). In recommendation systems, collaborative filtering combines known preferences of a group of users to predict preferences for new similar users or unrated items (e.g., [Adomavicius and Tuzhilin 2005]). Next, we outline some of the proposed methods for rank aggregation and how they can be applied to compose preferences.

In the context of social choice, Condorcet [1785] outlines a generic method designed to simulate pair-wise elections in a voting system. The criterion for selecting an item *x* as a winner, is that, for all other items *y*, *x* is preferred over *y* by the majority. In the context of preferences, for example, this criterion can be used to compose preference relations qualitatively, as follows:

DEFINITION 19. *Let S_P be a set of preference relations defined over the same relational schema R. The majority aggregation preference relation $\succ_{mp}$ is defined over R, such that $\forall\ t_i, t_j \in R$, let $P_1 \subseteq S_P$ be the set of preference relations P_x such that $t_i \succ_{P_x} t_j$, and $P_2 \subseteq S_P$ be the set of preference relations P_y such that $\neg(t_i \succ_{P_y} t_j)$, then $t_i \succ_{mp} t_j$, if and only if, $|P_1| > |P_2|$.*

Another example from voting theory is the approach proposed by Borda [1781]. In preference composition terms, it suggests a combining function that assigns as a final score to each item the sum of its positions in the initial *m* rankings. Yet another proposal from voting theory is to rank each tuple *t* based on both the number of tuples that *t* dominates and the number of tuples *t* is dominated by. This is also known as the Copeland index.

Masthoff [2004] reviews different aggregation strategies or functions for group preference modeling. Group preference modeling has been recently discussed by Amer-Yahia et al. [2009], in the context of group recommendations. Given a set of rankings representing recommendations for individual users, group recommendations are computed by aggregating the rankings using a *consensus function* that takes into account both the tuples degrees of interest for the users and the level at which users disagree with each other. Different strategies are used, such as the least misery, where the idea is that a group is as happy as its least happy member.

The rank aggregation problem can also be formalized as:

DEFINITION 20. *Given m ordered list of items, find an ordered list σ of the union of the items appearing in all m lists such that σ has the minimum total distance from the m lists.*

A list is called *full* if it includes all items. Well-studied distance metrics between two full ordered lists include the *Spearman footrule distance* that measures the sum of the differences in positions of the items in the two lists, and the *Kendall tau distance* that counts the pair-wise disagreements between the two lists. The aggregation obtained by optimizing the Kendall distance, called *Kemeny optimal aggregation*, corresponds to the geometric median of the input lists. Computing this aggregation was shown to be NP-hard and appropriate heuristics have been developed in the context of web search for reducing web spam based on *local Kemenization* [Dwork et al. 2001]. These heuristics essentially push Condorcet loosers to the bottom of the final ranking. It was also shown that a good heuristic for the footrule optimal aggregation is to sort all items by the median of their positions in all *m* ranks. This is actually optimal if all median positions are distinct. The optimal distance-based formulation of the aggregation problem is considered for the case of ranking with ties in [Fagin et al. 2006; 2004; Fagin et al. 2003]. Appropriate metrics are proposed that generalize the Spearman footrule and the Kendall tau distances by either breaking ties in all possible ways or summarizing the rankings into compact vectors. Efficient polynomial time algorithms are presented for the computation of all proposed metrics.

A majority-based approach is taken in [Cohen et al. 1999] in the context of web. The input in this case is a set of binary preference relations and the goal is to find a total order of all items that maximizes the number of pair-wise agreements. The problem is shown to be NP-complete and a greedy heuristic is proposed that approximates this optimal total order by assigning to each item a score based on the Copeland index.

3.4 Combining Preferences of Different Granularity

The mechanisms presented in the previous sections compose preferences for tuples. In this section, we study composition of preferences of different granularity.

In the preference model of Koutrika and Ioannidis [2005b], user preferences are expressed at the tuple and relationship level. Recall that in this model, preferences are expressed as (*Condition, Score*) pairs where *Condition* is a logical formula and *Score* is the score assigned to tuples satisfying *Condition* (Definition 4). Two preferences P_x and P_y are *composeable*, if and only if: (*i*) P_x is a join preference of the form (q_x, d_x) connecting R_x to a relation R_y, and (*ii*) P_y is a join or selection preference (q_y, d_y) on R_y.

DEFINITION 21. *An* implicit preference (q, d) *defined from two composeable preferences P_x and P_y, is a preference on $R \equiv R_x$, such that: (i) q is the conjunction of the conditions q_x and q_y and (ii) d is a score which is the result of a combining function f of the degrees of interest of the two preferences, i.e., $d = f(d_x, d_y)$.*

The combining function f used to compute d is usually a non-increasing function, such as the product or the minimum of the degrees of interest d_x and d_y, on the ground that it cannot exceed the degrees of interest of its supporting preferences.

Example 17: Addison likes actress J. Roberts, captured by the selection preference: (*actor.name = 'J. Roberts'*, 0.8). Moreover, she has expressed specific preferences over joins between the relations of the database schema (Figure 1). In particular,

Table III. Preference composition w.r.t. attitude.

	Attitude	
	Overriding	Combinatory
Qualitative	prioritized, lexicographical	Pareto, multidimensional Pareto, intersection, difference, union, disjoint union, linear sum, majority aggregation
Quantitative	–	*max*, *min*, *avg*, *weighted average*

she considers the actor of a movie to be very important and hence, she has provided the join preferences: (*movie.mid* = *play.mid,* 0.9) and (*play.aid* = *actor.aid,* 1). We can define an implicit preference for movies with J. Roberts, by for example, taking as a final score the product of the degrees of interests, as follows: (*movie.mid* = *play.mid* and *play.aid* = *actor.aid* and *actor.name* = '*J. Roberts*', 0.72).

Georgiadis et al. [2008] define preferences between both (*i*) values of specific attributes (tuple level) and (*ii*) attributes themselves (attribute level). Attribute level preferences are used to select the appropriate mechanisms for composing tuple preferences. The following example illustrates this.

Example 18: Consider the movie instance of Figure 2 and the preferences: (*i*) A. Hitchcock is preferred over M. Curtiz or S. Spielberg (preference P_D), (*ii*) horror movies are preferred over drama movies (preference P_G) and (*iii*) the director of a movie is equally important with its genre (preference P_{DG}). To combine preferences P_D and P_G, the proposed model selects the Pareto preference composition $P_D \otimes P_G$, since, as expressed by the third preference, P_D and P_G are equally important. Then, with regards to $P_D \otimes P_G$, t_2 is preferred to t_1 and t_3, and t_1, t_3 are incomparable.

3.5 Summary of Preference Composition

Given a set of preferences, preference composition seeks to combine them. The preferences to be composed may correspond to variant interests of a single user or to preferences of a group of users that need to be aggregated to express the group as a unit. With the increasing popularity of social networks, since users interests are in general diverse, composition is central for the success of personalization.

Table III summarizes composition mechanisms for tuple preferences. In general, composition reflects different user attitudes towards reconciling preferences and resolving conflicts. Compositions operators differ also on whether they preserve the properties of the preferences that they compose. Preserving specific properies is critical for providing sound theoretical formalisms. The resulting composed preference may be either expressed qualitatively or quantitatively. As discussed, any composition mechanism defined over preference relations can be also applied to preferences defined using functions (or degrees of interest). However, most approaches follow a pure qualitative or quantitative approach for preference representation and composition. For example, if one approach represents preferences using scoring functions then it also combines preferences using scoring functions. Besides mechanisms for composing tuple preferences, composition mechanisms for preferences of different granularity have also been considered, for example, for composing preferences for tuples with preferences for attributes. Although preferences could be used to express interest on specific composition approaches, this is an issue that is not fully explored. For example, this is implicitly achieved through preferences on attributes in [Georgiadis et al. 2008].

4. PROCESSING PREFERENCE QUERIES

Preferences are used in query processing to provide users with customized answers usually by changing the order and possibly the size of results. We organize methods for integrating preferences with queries into two general groups.

—*Query expansion methods*. These methods intend to personalize a given query by expanding it to include preferences, for example, by adding selection conditions that express user preferences on specific attribute values.

—*Methods with special preference operators*. These methods use special preference operators to explicitly specify preferences along with each query. Such preference operators are either implemented inside the database engine or translated to existing logical or physical operators.

We can further distinguish preference queries based on how preferences are integrated in a database query. In general, preferences are considered soft constraints as opposed to the selection conditions of the original query. This is the approach taken by most of the methods that use special preference operators. In contrast, in [Koutrika and Ioannidis 2005a; 2005b], the original query is expanded with additional conditions inferred by the user preferences, thus, in a sense, preferences are treated as hard constraints. Another distinction between preference queries can be based on whether they result on a qualitative or a quantitative ranking of the output. In the former case, the dominant tuples are returned, whereas in the latter, a score is associated with each tuple in the result.

In the rest of this section, we study first query processing techniques related to methods that expand database queries (Section 4.1) and then issues related to using special preference operators (Section 4.2), including top-*k* queries (Section 4.3). We also discuss methods for improving performance through pre-processing steps (Section 4.4).

4.1 Expanding Database Queries

In this case, the existence of a predefined set of user preferences, often called *user profile*, is assumed. This set is used to expand regular database queries with conditions that express such preferences. In doing so, there are three basic steps:

(1) *Determining preference relatedness*. The first step involves determining which preferences are related and hence, applicable to a given query.

(2) *Filtering related preferences*. In some cases not all related preferences should be used, and thus, an extra step is needed for identifying which of these preferences should be integrated into the query.

(3) *Preference query formulation*. Finally, the query is expanded with the selected preferences.

4.1.1 *Preference Relatedness.* There are many different ways of defining the relevance of a preference to a query. Let (C, P) be a preference, indicating that P is defined for context C, which can be internal, external (or both), or null if the preference is context-free (in which case, we could simply write P). Similarly, let (C_Q, Q) be a user query Q formulated over the database, where C_Q indicates the external context of Q and is null if no external context is specified. C_Q is formulated using the same set of context parameters that are used for the external part of C.

Definition 22. *A preference* (C, P) *is* related *to a query* (C_Q, Q) *if: (*i*) the external part of context* C matches *the external query context* C_Q *and the internal part of the context* C *(if any)* matches Q*, and (*ii*) the preference part* P *is* relevant to *(a subset of) the results of* Q.

4.1.1.1 *Context Matching.* To determine how well the context of a preference matches the context of a query, a measure of the distance, similarity or difference between contexts is used. This measure depends on the type of context. A commonly used approach is to adopt a vector representation for context and then apply known distance metrics.

In the case of internal context, the context C of a preference specifies conditions that the tuples of a database instance r must satisfy for P to hold. This context must match the query Q. As an example of a vector-based approach for this case, we consider the approach in [Agrawal et al. 2006]. Let $R = \{A_1, \ldots, A_d\}$ be a relational schema. In this model, a preference has the form (C, P), where C is a logical formula expressing conditions on the values of the attributes that must be satisfied for P to hold, specifically $C = \wedge_{j \in L}(A_j = a_j)$, $L \subseteq \{1, \ldots, d\}$ and $a_j \in dom(A_j)$. Q is a conjunctive query. Let $\mathcal{D}$ be the set of all N distinct $< attribute, value >$ pairs appearing in an instance r of R. We refer to the i-th element of $\mathcal{D}$ by $\mathcal{D}[i]$. A vector representation of C is a binary vector V_C of size N whose i-th element corresponds to $\mathcal{D}[i]$. If $\mathcal{D}[i]$ appears among the conjunctions of C, then $V_C[i] = 1$; otherwise it is 0. Analogously, the vector representation of Q is a binary vector V_Q of size N, where $V_Q[i] = 1$, if $\mathcal{D}[i]$ is one of the conjuncts in Q; otherwise it is 0. The similarity of C and Q is then defined using their vector representations V_C and V_Q as follows:

$$sim(C, Q) = cos(V_C, V_Q) = \frac{V_C \cdot V_Q}{|V_C||V_Q|}.$$

In the case of external context, the external context of a preference must match that of the query. A special case applies when the context parameters take values from hierarchical domains (e.g., the hierarchies in Figure 9). Then, it is possible to compare contexts expressed at different levels of abstraction using the notion of coverage [Stefanidis et al. 2007a]. For instance, we can relate a context in which the parameter *time_period* is instantiated to a specific occasion (e.g., *Christmas*) with a context in which the same parameter is expressed with a more general period (e.g., *holidays*). Based on coverage, we can relate the external context of a preference (C, P) to the context C_Q of a query, if C is more general than C_Q, that is, if the context values specified in C are equal to, or more general than the ones in C_Q [Stefanidis et al. 2007a; Miele et al. 2009]. This can be generalized as follows. A context C is related to a query context C_Q, if C is relaxing zero or more parameters of C_Q in any of the following ways [Stefanidis et al. 2007b]: *upwards* by replacing its value by a more general one, *downwards* by replacing its value by a set of more specific ones, or *sideways* by replacing its value by sibling values in the hierarchy. Given all these possible relaxations, appropriate distance metrics that exploit the number of relaxed parameters and the associated depth of such relaxations are employed to measure how well context C matches C_Q.

Bunningen et al. [2006] consider query expansion using external contextual preferences where contexts and preferences are defined through description logic concept expressions. Contextual preferences are considered relevant to a query if their

contexts are the same with or more general than the query context and their preferences contain concepts which can be mapped to certain relations of the query. Stefanidis et al. [2010] define a qualitative preference model with internal context for keyword search in relational databases. Both context and queries are sets of keywords. A preference is considered related to a query if its context is the same with or more general than the query (i.e., contains a subset of the query keywords).

4.1.1.2 *Preference Relevance.* In general, a preference is relevant to a query, if, when combined with a query, it yields an interesting, non-empty result. For example, consider a preference P on the genre of a movie and a query Q about actors born before 1900. Should P be used in evaluating Q perhaps by ranking the actors in the result of Q based on the genre of the movies in which each actor appears? Furthermore, in identifying relevant preferences, should we consider just the given set of preferences or also possible compositions of them? There is no single definition of relevance. Koutrika and Ioannidis [2004] consider both explicit and implicit (i.e., composed) preferences as relevant, if they map to a path on the database graph attached to a relation included in the query.

Applicability is a special form of relevance, when preferences are considered hard constraints. A preference P is *applicable* to a query Q, if the execution of Q combined conjunctively with P (over the current database instance) yields a non empty result. This may be also called *instance applicability*. For example, consider a query about recent movies and a preference for movies directed by S. Spielberg. This preference is instance applicable only if the database contains recent entries of this director. In general, instance applicability can only be checked by actually executing the query with the preference.

Another type of applicability is semantic applicability. A preference P is not *semantically applicable* to a query Q, if the execution of Q combined conjunctively with P over any database instance yields an empty result. To decide whether a preference is semantically applicable to a query, knowledge outside the database may be needed. As an example, take a query about comedies. Then, a preference for movies directed by A. Tarkovsky is not semantically applicable to this query, since this director has not directed any comedies. Note that if a preference P is instance applicable to a query Q, then, it is also semantically applicable to Q. The reverse does not always hold. In some special cases, the applicability of a preference P to a query Q can be determined simply by a syntactic analysis of P and Q. For example, a query about movies released after 2000 and a preference for movies released prior to 1990 are conflicting and will return an empty result when combined through a conjunction. On the other hand, a preference for movies with actor B. Stiller is syntactically applicable to a query for movies with J. Roberts. Note that a preference P that is syntactically applicable to Q, it is not necessarily instance applicable. However, the reverse always hold.

4.1.2 *Preference Filtering.* All preferences related to a query may be used for ranking and selecting the tuples returned by the query. Alternatively, preferences can be ranked based on their preference score (showing their intensity) or their relatedness score (capturing the degree to which a preference is related to a query based on some context matching function, their relevance to query or both). Then, the best of them are selected. Note that selecting too many preferences may lead to over-restricting the result, while selecting too few may not suffice to express the

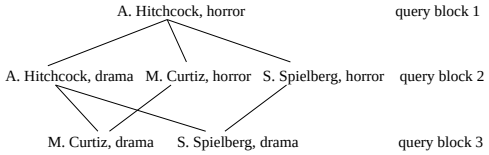


Fig. 12. Query lattice example.

initial user intent.

Koutrika and Ioannidis [2004] propose an algorithm for selecting at query time the top K (explicit and implicit) preferences based on their preference score. Preference filtering has been also modeled as an optimization problem with constraints [Koutrika and Ioannidis 2005a]. The parameters of the problem are the execution cost of the query, the size of its result and the degree of interest of its tuples. The objective is to select a set of preferences that, in conjunction with the query, will optimize one of the parameters and satisfy constraints on the others. A state space search approach is used and a number of algorithms are proposed that find the optimal subset of related preferences that matches the problem constraints.

4.1.3 *Formulation of Preference Queries.* Once the set of related preferences $\mathcal{P}$ is selected, the preferences in $\mathcal{P}$ are used to rewrite the original query Q to construct one or more preference queries. Depending on the preference model and the desired result, one may think of different ways to integrate the preferences with Q. Next, we outline two approaches.

Koutrika and Ioannidis [2004] uses the top K preferences to modify Q, a process termed *query personalization*, and generate results that satisfy at least L of the K preferences. Parameter K determines the desired extent of personalization, while parameter L captures the minimum number of user criteria (i.e., preferences) that an answer should meet. Two different query re-writing mechanisms are possible. The first one composes a single query that defines a conjunction of the initial query Q with the disjunction of all possible conjunctions of the L from the K preferences. In the second mechanism, at most K queries are formulated, each one augmenting Q with one of the K preferences. The partial results are grouped and each tuple that appears at least L times is output. In both cases, a ranked list of results is returned according to the preferences they satisfy.

Georgiadis et al. [2008] consider a different but related problem: given a relation R and a set $\mathcal{P}$ of preferences defined over R, how to formulate preference queries that would produce as output a ranking of R based on $\mathcal{P}$. To this end, they build a query lattice with one node for each combination of values of different attributes appearing in $\mathcal{P}$. For example, for preferences: (*i*) A. Hitchcock is preferred over M. Curtiz or S. Spielberg, (*ii*) horror movies are preferred over drama movies and (*iii*) the director of a movie is as important as its genre, the query lattice of Figure 12 is constructed. A query is formulated for each node in the lattice. All queries in a specific block produce equally preferable results. The queries of each block are successively executed starting from the queries of the top block and going down the lattice. For example, for the instance shown in Figure 2, t_2 is the result of the query of the first block, the queries of the second block return no results and the third block returns tuples t_1 and t_3.

4.2 Using Preference Operators

Another way of integrating preferences in querying relational databases is by extending query languages with preference-related operators. Next, we first present a number of such operators and then discuss their implementation.

4.2.1 *Types of Preference Operators.* The most basic preference operator is one that selects from its input the set of the most preferred tuples. This operator is called winnow [Chomicki 2003], Best [Torlone and Ciaccia 2002] and preference selection operator [Kießling 2002]. Formally,

DEFINITION 23. *Given an instance r of a relational schema R and a preference relation P over R, the winnow operator* $win_P(r)$ *is defined as:*

$$win_P(r) = \{t_i \in r \mid \nexists t_j \in r, \text{ such that } t_j \succ_P t_i\}.$$

A database tuple t_i belongs to the winnow if it is not "killed" or *dominated* by another tuple t_j, that is, if no other tuple t_j in r is preferred over t_i. Winnow can be used to select tuples for more than one relation when applied to the result of queries defined over more than one relation.

Clearly, $win_P(r) \subseteq r$. It has been shown [Chomicki 2003], that for every finite, non-empty instance r of R, if $\succ_P$ is a strict partial order, then $win_P(r)$ is non-empty. For any two tuples t_i and t_j of r that belong to $win_P(r)$, it holds that $t_i \sim_P t_j$, that is, they are indifferent. When the preference relation $\succ_P$ is a total order, $win_P(r)$ includes just one tuple, whereas when the preference relation $\succ_P$ is a weak order, the tuples in $win_P(r)$ are the tuples that belong to the top equivalence class of r defined by $\sim_P$. Partial antimonotonicity also holds when $\succ_P$ is a strict partial order: $\forall\, r_1, r_2$ of R, $r_1 \subseteq r_2 \Rightarrow win_P(r_1) \supseteq win_P(r_2) \cap r_1$ [Chomicki 2003].

Kießling [2002] defines two relational operators: (*i*) the *preference selection operator* and (*ii*) the *grouped preference selection operator*. The preference selection operator, denoted $\sigma[P](r)$, corresponds to the winnow operator $win_P(r)$. The grouped preference selection operator applies preference selection within groups. Given a subset B of the attribute set of R, tuples in r are partitioned into groups of tuples having the same values in the grouping attributes in B. The grouped preference selection operator $\sigma[P \text{ group by } B](r)$ selects the dominating tuples in each group. Formally:

DEFINITION 24. *Given an instance r of a relational schema R and a preference relation P over R, the grouped preference selection operator* $\sigma[P \text{ group by } B](r)$ *is defined as:* $\sigma[P \text{ group by } B](r) = \{t_i \in r \mid \nexists t_j \in r, \text{ such that, } t_j \succ_P t_i \wedge t_i[B] = t_j[B]\}$, *where B is a subset of the attribute set of R.*

A popular form of incorporating preferences in SQL is by applying a *skyline operator* to pick from the query result only those tuples that are not dominated by any other tuple in the result, where dominance is based on Pareto composition semantics. A tuple t_i dominates another tuple t_j, if t_i is as good as or better than t_j with regard to a set of preferences and better than t_j with regard to at least one preference.

Typically, skylines are formulated in multidimensional Euclidean spaces where the preference relation is either $>$ or $<$. Let $A = \{A_1, A_2, \ldots, A_d\}$ be a set of d attributes (also called dimensions). Let $dom(A) = dom(A_1) \times \ldots \times dom(A_d)$ and the preference relation be $>$. Let $t_i, t_j \in dom(A)$, t_i dominates t_j, $t_i \succ_D t_j$, if and only if, for all attributes A_m, $1 \leq m \leq d$, $t_i[A_m] \geq t_j[A_m]$ and for some l, $1 \leq l \leq d$. $t_i[A_l] > t_j[A_l]$. A tuple belongs to the skyline, if and only if, it is not dominated by any

other point. Clearly, the skyline operator corresponds to the winnow operator.

Skylines were first introduced in [Börzsönyi et al. 2001] for multidimensional Euclidean spaces using the following clause:

SKYLINE OF A_1 [MIN | MAX | DIFF], . . ., A_m [MIN | MAX | DIFF]

where a MIN (or MAX) specification following an attribute or dimension A_i expresses preference of small (or large) values of A_i and DIFF that only tuples with identical values of A_i are comparable.

There are several recent proposals for refining the typical skyline dominance definition. Chan et al. [2006] define the *k*-dominant skyline: a tuple t_i *k*-dominates another tuple t_j if there are *k* dimensions, or preferences, in which t_i is better than or equal to t_j, and t_i is better in at least one of these *k* dimensions. Lin et al. [2007] propose the *k*-representative skyline that selects *k* tuples, such that, the number of tuples that are dominated by at least one of these *k* tuples is maximized, while Xia et al. [2008] introduce the ϵ-skyline that computes the set of all tuples that are not ϵ-dominated by any other tuple. Given a set of preferences, a tuple ϵ-dominates another tuple if it is as good, better or slightly worse (up to ϵ) with regard to all preferences and better in at least one preference.

Both the winnow and the skyline operators select the most preferred tuples from a given input set. Ranking the whole input can be achieved by repetitive applications of such operators. The iterated winnow operator is formally defined as follows [Chomicki 2003; Torlone and Ciaccia 2003]:

DEFINITION 25. *Given an instance r of a relational schema R and a preference relation P over R, the iterated winnow operator, $win_P^i(r)$, of level i, i > 0, is defined as follows:*
—$win_P^1(r) = win_P(r)$
—$win_P^{i+1}(r) = win_P(r - \cup_{k=1}^{i} win_P^k(r))$

Tuples retrieved earlier are of higher interest to the users. All tuples in any $win_P^i(r)$ are indifferent to each other.

Another commonly used preference operator in SQL is top-*k* expressed by adding a clause of the form:

ORDER BY [user-defined-function] STOP AFTER *k*

In this case, the tuples in the result are ranked according to a user defined scoring function and the results with the *k* highest scores are returned. Note that repetitive applications of *skyline* correspond to a qualitative ordering of results, whereas an application of *top-k* to a quantitative one.

Besides skyline and top-*k*, there are two rather comprehensive proposals of extending SQL with preference operators, namely, *Preference SQL* [Kießling and Köstler 2002] and recently *FlexPref* [Levandoski et al. 2010].

Preference SQL is an extension of SQL that covers all base preference constructors of [Kießling 2002]. Preferences can be defined over several relations. It supports both hard constraints in the WHERE clause and soft constraints using a special PREFERRING clause:

PREFERRING [conditions] BUT ONLY [conditions]

Preference SQL follows a "best matches only" semantics. First, it finds all perfect matches to the conditions of PREFERRING. If this set is empty, then it considers all other best matching values excluding those that do not satisfy the BUT ONLY quality conditions. For example, the query:

PREFERRING year AROUND 2010 BUT ONLY distance(year)<= 2

express a preference for movies of 2010, but if such movies do not exist, the user is willing to accept variations of at most two years.

The goal of *FlexPref* [Levandoski et al. 2010] is to provide a framework for the integration of any preference operator inside the database engine. It adds the following clauses to SQL:

PREFERRING [preference attributes] USING [method]
WITH [parameter] OBJECTIVES [objective]

The USING clause specifies the preference method (such as skyline or top-*k*) with objectives (such as MIN for the case of skyline) applied over the attributes defined in the PREFERRING clause WITH specific values for the parameters (such as parameter *k* for top-*k*). For example, the query:

SELECT * FROM movies USING skyline
WITH OBJECTIVES MIN duration, MAX year

evaluates the skyline on movies, where the preference objective requires minimizing duration, while maximizing the year of release.

4.2.2 *Implementation of Preference Operators.* There are the following general ways of handling preference operators.

—Preference operators can be implemented on-top of the DBMS either as stand-alone programs or as user-defined functions.
—Preference operators may be translated into other, existing relational algebra operators during a pre-processing step.
—Preference operators can be implemented inside the database engine using specific physical operators and algorithms.

We present first specialized algorithms for implementing winnow. These can be implemented within the database engine or on top of the DBMS on the returned query results. The naive way to compute the winnow of a relation instance r is to apply a basic *Nested-Loop* (NL) method that compares each tuple in r with every other tuple. The NL method works for every type of preference relation $\succ_P$ but requires scanning the whole r for each tuple, which becomes inefficient for large r.

A more efficient implementation is the *Block-Nested-Loop* (BNL) algorithm proposed by Börzsönyi et al. [2001] in the context of skyline queries. BNL maintains a window W of indifferent tuples, which comprise the best tuples found so far. At each iteration, all tuples in the input are read. When a tuple t is read, it is compared with all tuples in W. If t is dominated by a tuple in W, then t is discarded. If t dominates one or more of the tuples in W, these tuples are discarded and t is inserted into W. Finally, if t is indifferent with all tuples in W, t is inserted into W. At the end of each iteration, all tuples added to W are output since these are the most preferred ones. BNL uses transitivity and works correctly only when the preference relation $\succ_P$ is at least a strict partial order. To see this, say that an input tuple t_j is dominated by a tuple t_i in W, thus t_j is discarded. Next, a tuple t_k arrives that is dominated by t_j, but not by t_i in W. The algorithm may output t_k incorrectly.

When $\succ_P$ is a weak order, the *Winnow for Weak Orders* (WWO) algorithm takes advantage of the fact that all tuples in the winnow belong to a single equivalence class and can further improve on BNL [Chomicki 2007b]. A single comparison of t with just one tuple in W suffices to determine whether t will be added in W or will replace the whole W or be discarded. At the end of the first iteration, W will contain only tuples in the winnow.

The *Sort-Filter-Skyline* (SFS) algorithm for computing skylines [Chomicki et al. 2003] adds a pre-processing step to BNL that sorts all tuples in r, so that, if a tuple $t_i \succ_P t_j$, then t_i precedes t_j in the produced order. This corresponds to the order produced by a topological sort of the preference graph of r. By processing the tuples following this order, it is ensured that when a tuple is inserted into the window W, it belongs to the winnow, thus it can be output immediately. For SFS to work, $\succ_P$ must be at least a strict partial order. The *LESS* algorithm refines SFS by eliminating non-skyline tuples during the sort process [Godfrey et al. 2007]. SaLSa performs also a topological sort of the input [Bartolini et al. 2008]. Sorting is used as a means to stop fetching tuples from the input stream, thus limiting the number of tuples to be read.

Morse et al. [2007] and Preisinger and Kießling [2007] proposed algorithms for low cardinality domains with linear worst-case complexities. They are based on a lattice data structure that captures the dominance relationship between equivalence classes.

Several other approaches have been proposed for computing skylines but their detailed presentation is beyond the scope of this survey. A large class of these approaches use indexing. For example, Tan et al. [2001] introduced the first progressive algorithm that returns skylines without scanning the whole dataset using a Bitmap and an index algorithm. Kossmann et al. [2002] proposed another progressive algorithm based on the nearest neighbor search method, while Papadias et al. [2003] introduced a branch-and-bound algorithm where datasets are indexed by an R-tree. Skyline computation in subspaces has been studied e.g., in [Yuan et al. 2005; Pei et al. 2005; Tao et al. 2006]. Zhang et al. [2009] exploits an index structure to improve the performance of sort-based approaches. The focus is on reducing the computation cost, instead of the I/O cost. This is achieved by determining appropriate partitions of the search space that reduce the number of comparisons between the input tuples and the current skyline points.

A straightforward implementation of the iterated winnow operator can be achieved by applying one of the previous algorithms (e.g., the NL or SFS) multiple times. The first application would be on instance r to produce $win_P^1(r)$ and the subsequent applications on $(r - \cup_{k=1}^{i} win_P^k(r))$, to produce $win_P^{i+1}(r)$. A more efficient implementation of winnow and ranking is proposed in [Torlone and Ciaccia 2002; 2003]. The *Evaluating Best Operator Algorithm* is a variation of BNL, where the repetition for computing $win_P^{i+1}(r)$ does not start from scratch each time, but instead, from those tuples that were found to be directly dominated by a tuple in $win_P^i(r)$. The iterated winnow operator can also be implemented by topologically sorting the preference graph of r [Drosou et al. 2009; Georgiadis et al. 2008].

Table IV depicts the worst-case time complexities and the preference relation requirements of the basic algorithms for evaluating winnow. In general, these algorithms do not require indexing or pre-processing and present quadratic worst-case and linear average-case with regard to the size of the input relation. In Table V, we present approaches that employ indices for computing skylines also indicating which ones are progressive. An analysis of the run time complexity of many skyline related algorithms can be found in [Godfrey et al. 2007].

Table IV. Time complexities and preference relation requirements of winnow algorithms.

Algorithms	Worst-case complexity	Preference relation
NL	$O(n^2)$	*any*
BNL [Börzsönyi et al. 2001]	$O(n^2)$	*strict partial order*
WWO [Chomicki 2007b]	$O(n)$	*weak order*
SFS [Chomicki et al. 2003]	$O(n^2)$	*strict partial order*
LESS [Godfrey et al. 2007]	$O(n^2)$	*any*
SaLSa [Bartolini et al. 2008]	$O(n^2)$	*any*
Best [Torlone and Ciaccia 2002]	$O(n^2)$	*any*
[Drosou et al. 2009]	$O(n^2)$	*strict partial order*
LBA, TBA [Georgiadis et al. 2008]	$O(n)$, $O(n^2)$	*preorder*

Table V. Use of indices for computing skylines and progressive computation.

Algorithms	Index	Progressive
Index [Tan et al. 2001]	B+-tree	✓
Nearest Neighbor [Kossmann et al. 2002]	R-tree	✓
Branch-and-Bound [Papadias et al. 2003]	R-tree	✓
Subsky [Tao et al. 2006]	B-tree	
OSP [Zhang et al. 2009]	LCRS-tree	✓

In the case of implementing winnow inside the database engine, a number of interesting research problems are introduced regarding the semantic optimization of relational queries that include winnow operators. For semantic optimization, a set of algebraic rules that characterize the interaction of winnow with the standard relational operators, such as commutativity, are provided in [Chomicki 2003]. These rules can be used to optimize a query, for example by pushing selections and projections down the query tree. Further optimizations in the presence of integrity constraints, such as eliminating redundant applications of winnow, are possible [Chomicki 2007b]. Such optimizations are achieved by relativizing the properties of the given preference relations to the sets of instances that satisfy the integrity constraints of the particular database. Additional semantic optimization techniques are developed by Endres and Kießling [2008].

Apart from using special evaluation and optimization algorithms to implement preference operators, it is possible to translate preference operators to existing relational algebra operators. For example, winnow can be translated to existing relational algebra operators [Chomicki 2003; Kießling 2002]. This is the approach taken to implement Preference SQL, where preference queries expressed using such operators are translated into standard SQL queries [Kießling and Köstler 2002].

Note that Hafenrichter and Kießling [2005] offer also a hybrid approach to implementing Preference SQL that provides support for preference query processing within the database engine. The query optimizer is extended to include transformation rules for the preference selection operator (i.e., winnow) and the group preference selection operator.

Finally, a different approach to implementing preference operators is taken by *FlexPref* [Levandoski et al. 2010]. The goal is to facilitate the integration of any preference operator into the database engine with minimal effort. A preference method can be integrated by registering a set of functions that define rules for the following: (*i*) given two tuples, when one is preferred over the other and (*ii*) given

a tuple t and a set of preferred tuples whether t is a preferred tuple and can be added to the set. Once a preference method is injected into the database, it can be used for querying the database.

4.3 Top-k Query Processing

Typically, a top-k query aims at providing only the top k most important results to the users. A common way to identify the k most important results is scoring all tuples based on a scoring function, possibly defined as an aggregation of a set of functions over different attributes, and reporting the k tuples with the highest scores. Although there is a large amount of research that addresses top-k processing techniques, we consider that the details of such techniques are out of the scope of this survey, since in most cases, they are not directly related to user preferences. Thus, below, we provide only an overview of the main approaches. A survey of top-k processing techniques is presented by Ilyas et al. [2008].

In general, methods for compounding a set of rankings to an aggregate one consider that each tuple in each ranking is associated with an interest score that determines its position within the ranking. Then, to construct a total ranking, instead of following the naïve approach of computing the aggregate score of each tuple and ranking the tuples based on these scores, several more efficient algorithms have been proposed.

A fundamental algorithm, called *FA* algorithm, for retrieving the top-k tuples of a relational schema R is proposed by Fagin et al. [2001]. This algorithm considers two types of available tuple accesses: the *sorted* access and the *random* access. Sorted access enables tuple retrieval in a descending order of their scores, while random access enables retrieving the score of a specific tuple in one access. Next, we present the main steps of the *FA* algorithm.

—First, do sorted access to each ranking until there is a set of k tuples, such that each of these tuples has been seen in each of the rankings.
—Then, for each tuple that has been seen, do random accesses to retrieve the missing scores.
—Compute the aggregate score of each tuple that has been seen.
—Finally, rank the tuples based on their aggregate scores and select the top-k ones.

FA is correct when the aggregate tuple scores are obtained by combining their individual scores using a monotone function. This also holds for the *TA* algorithm [Fagin et al. 2001]. *TA* ensures further that its stopping condition always occurs at least as early as the stopping condition of *FA*. Its main steps are the following:

—First, do sorted access to each ranking. For each tuple seen, do random accesses to the other rankings to retrieve the missing tuple scores.
—Then, compute the aggregate score of each tuple that has been seen. Rank the tuples based on their aggregate scores and select the top-k ones.
—Stop to do sorted accesses when the aggregate scores of the k tuples are at least equal to a threshold value that is defined as the aggregate score of the scores of the last tuples seen in each ranking.

Nepal and Ramakrishna [1999] and Güntzer et al. [2000] independently propose algorithms equivalent to the *TA* algorithm. Several *TA* modifications with regard to the access type that can be applied have been introduced (e.g., [Fagin et al. 2001; Güntzer et al. 2001]). For example, the *NRA* algorithm is appropriate when random accesses are expensive or impossible and so, only sorted accesses are employed,

Table VI. A taxonomy of top-*k* query processing techniques.

		Implementation Level	
		Application Level	Within Engine
Query Model	Top-*k* Tuples	[Fagin 1999; Fagin et al. 2001; Nepal and Ramakrishna 1999; Güntzer et al. 2000; 2001]	––
	Top-*k* Join Tuples	[Natsev et al. 2001]	[Ilyas et al. 2004]
	Top-*k* Groups of Tuples	––	[Li et al. 2006]

while the *CA* algorithm is appropriate when random accesses are expensive relative to sorted accesses [Fagin et al. 2001]. Balke et al. [2002] propose the *SR-Combine* algorithm for top-k query processing in mobile environments.

All the above approaches focus on constructing an aggregate ranking by combining a set of rankings that contain the same set of tuples. Clearly, in this case, the produced ranking consists of the same tuple set. Apart from such approaches, there are algorithms (e.g., [Natsev et al. 2001; Ilyas et al. 2004]) for aggregating rankings that contain different sets of tuples. In this category of algorithms, tuples of different rankings are joined together with respect to specific join conditions. The produced ranking here consists of a set of joined tuples, each one with an aggregate score computed from the scores of the participating tuples. Instead of working on individual tuples, Li et al. [2006] propose a method that reports the *k* groups of tuples with the largest interest scores, where scores are computed using a group aggregation function, such as *sum*.

In Table VI, we organize the main top-*k* query processing techniques with respect to: (*i*) the *query model* and (*ii*) the *implementation level*. The query model defines the kind of results returned by the top-*k* computation, that is, single tuples (e.g., [Fagin 1999; Nepal and Ramakrishna 1999]), joined tuples (e.g., [Natsev et al. 2001; Ilyas et al. 2004]) or groups of tuples (e.g., [Li et al. 2006]). The implementation level defines the level of integration with database systems, that is, at application level outside the database engine (e.g., [Fagin et al. 2001; Nepal and Ramakrishna 1999; Güntzer et al. 2000]), or within the query engine, such as the approach proposed by Ilyas et al. [2004] that define physical, nonblocking query operators based on variants of ripple join, that can be integrated into pipelined execution plans.

4.4 Pre-computation of Rankings

To avoid ranking large datasets at query time, pre-processing steps can be exploited for making online query processing faster. Most related approaches use preferences to compute and store (i.e., materialize) representative rankings of the database tuples offline. At query time, materialized rankings that are related to the query are selected and possibly combined to generate the final results. We organize existing approaches to answering preference queries as:

—*Context-based* approaches, where preferences hold under specific conditions.
—*Context-free* approaches, where preferences hold under all circumstances.

4.4.1 *Context-based Approaches.* Given a set of contextual preferences, Agrawal et al. [2006] initially produce a ranking for each set of preferences with the same context. Let T_m be this set of rankings. Since this set may be large, instead of materializing all such rankings, a smaller set, say T_l, of representative rankings is selected to be materialized. The selection is such that to minimize the cost of computing any τ in T_m from T_l defined as $\sum_{\tau \in T_m} d(\tau, T_l)$, where the distance between

a single ranking τ and a set of rankings T is defined as $d(\tau, T) = min_{\rho \in T} d(\tau, \rho)$ and the distance between two rankings is either the *Spearman footrule* or the *Kendall tau* distance. Since the selection of T_l is an NP-hard problem, different heuristics are proposed. For example, given the set of all rankings T_m, the *Greedy Algorithm* removes in each iteration the ranking which when removed, causes the least increase in the cost. The *Furthest Algorithm* starts by selecting randomly a ranking and at each step picks from the unselected rankings the one which is furthest from the already selected rankings. You and Hwang [2008] adopt the preference model proposed by Agrawal et al. [2006] and use a machine learning approach to induce a contextual total ranking from a partial quantitative ranking used as training set.

A different approach for pre-computing representative rankings is to create groups of similar preferences and produce a ranking for each group. Two different ways of defining similarity of contextual preferences are proposed by Stefanidis and Pitoura [2008]. The first one considers as similar the preferences that have similar contexts and groups them using a typical hierarchical agglomerative clustering method. Context similarity exploits the hierarchical nature of the domain of the context parameters. For each produced cluster, the resulting cluster description is selected as the representative context. A complementary method for grouping preferences is based on identifying those preferences that result in similar scores for all database tuples. This method exploits the quantitative nature of preferences and groups together preferences that have similar predicates and scores. Preferences are represented through a predicate bitmap matrix whose size depends on the desired precision of the resulting scores.

In both [Agrawal et al. 2006] and [Stefanidis and Pitoura 2008], when a user poses a query, the query is matched against the representative contexts, and those similar to the query are used for computing the query results.

4.4.2 *Context-free Approaches.* In a context-free scenario, there are approaches for computing the result of a preference query based on a set of materialized rankings constructed offline and maintained independently of specific conditions or circumstances. Usually, such approaches employ materialized preference views, that is, relational views ordered according to a preference, or scoring, function, to compute preferential query results.

The PREFER system provides ranked answers to preference queries based on a number of pre-computed and materialized views [Hristidis and Papakonstantinou 2004]. Given a relational schema $R(A_1, \ldots, A_d)$, a view v ranks the tuples of R with regard to a scoring function F_v defined as a weighted sum using a vector of weights $(w_1, \ldots, w_d)$. During the online phase, for each query Q that is also expressed via a weight vector, the view that best matches Q is selected. The view selection problem is formulated as the problem of identifying the view that needs the least number of tuples to be fetched for computing the query answer.

After locating the appropriate view, PREFER returns in a pipelined way, the k tuples that maximize the query preference function. In particular, the employed algorithm computes the smallest prefix of the view, i.e., the smallest number of top ranked tuples, to find the most preferred tuple for the query. Then, it computes a second prefix to find the most preferred tuple after the previous one and so forth, until the k most preferred tuples are retrieved. The key concept of this algorithm

is the computation of a watermark value which calibrates the stopping condition in each iteration of the algorithm. Such a watermark value determines how deep in the ranked materialized view we should go to locate the top tuple of a query. The first watermark for a view v is the maximum value $T^1_{v,Q}$, such that, $\forall t \in R$, $F_v(t) < T^1_{v,Q} \Rightarrow F_Q(t) < F_Q(t^1_v)$, where t^1_v is the tuple in v with the highest score. Respectively, at the second iteration of the algorithm, the tuple t^2_v, which is the tuple with the next higher score, replaces the tuple t^1_v in the process of watermark computation and so on.

Another view-based technique that also maintains ordered views based on preference functions is proposed by Das et al. [2006a]. This work utilizes the given set of views to produce query answers, using a linear programming algorithm. Yi et al. [2003] focus on the reduction of the maintenance cost of the materialized top-k views, considering occurrences of deletions and updates.

4.5 Summary of Processing Preference Queries

In general, preference queries exploit user preferences to rank their results and present to users the best among them. There are two general lines of work: (*i*) performing query expansion and (*ii*) using preference operators.

Query expansion methods assume the existence of a set of user preferences and integrate them into a regular query to personalize it. Issues related to this line of work include: (1) how to find the best set of preferences to use and (2) how to re-write the original query to integrate the selected preferences. There are many open problems with regards to both issues. For instance, there is the issue of over-personalization if too many preferences are selected or the issue of conflict resolution when conflicting preferences are applicable. Furthermore, performance-related issues with regards to the integration of preferences with query processing have been hardly explored.

Preference operators are used as additional clauses to a query to specify a way of ranking its results. Two widely used such operators are: (*i*) top-k, that ranks the results based on some user-defined scoring function and returns the k highest ranked ones, and (*ii*) skyline that returns the non-dominated results with regards to a Pareto composition of preferences on the values of the attributes of the tuples in the result. There is a large body of research on algorithms for computing them mostly to be used on top of the database engine either as stand-alone programs or as database user-defined functions. However, besides these two popular operators, there is a variety of other ways of expressing preferences. Nevertheless, in terms of supporting a general preference query model, the only comprehensive approach is Preference SQL. Another recent attempt is FlexPref, which aims at providing a flexible mechanism for the integration of any preference operator inside the database engine. This is a very promising topic.

5. CONCLUSIONS

Although the primary focus of this survey is on representation, composition and application of preferences in databases, we present in this section a short overview of some representative approaches to preference learning and revising, since they are interesting and active related areas of research with potential applications in data management (Section 5.1). We also present other preference models and

applications and discuss connections with other fields in which preferences are investigated. Finally, we highlight critical open research challenges and directions for future work (Section 5.2).

5.1 Other Issues

5.1.1 *Preference Learning.* Learning and predicting preferences in an automatic way has attracted much current attention in the areas of machine learning, knowledge discovery and artificial intelligence. Most methods for preference learning utilize information of the past user interactions in the form of a history or log of transactions. Joachims [2002] utilizes as input logs of the form of user clickthrough data, namely the query-logs of a search engine along with the log of links that the user actually clicked on from those in the presented ranked list. The fact that a user clicked on a link l_i and did not click on a link l_j ranked higher in the list than l_i, is interpreted as a user preference of l_i over l_j. Clikthrough data are used as training data to learn a ranking function that, for each query q, produces an order of links based on their relevance to q. This is achieved by selecting the function that produces the order having the minimal distance from the orders inferred from the clickthrough data. A support vector machine (SVM) algorithm is used.

User logs are also used as input by Holland et al. [2003], but in the form of relational instances. Since there is no explicit ranking information in the log of relations, to detect preferences, the frequencies of the different attribute values, i.e., their number of entries, in the log relation are used. Then, x is preferred over y, if and only if, $freq(x) > freq(y)$. Preferences between values of individual attributes are used to infer positive and negative preferences, numerical preferences and complex preferences [Kießling 2002].

The approach proposed by Cohen et al. [1999] is an example of employing user feedback to improve preference learning. They consider the problem of learning how to rank items given the feedback that an item should be ranked higher than another. For a set of items $\mathcal{I}$, the employed preference function $Pref(i_1, i_2)$, $Pref$: $\mathcal{I} \times \mathcal{I} \rightarrow [0, 1]$, returns a value indicating which item, i_1 or i_2, is ranked higher. The learning phase of such a function takes place in a sequence of rounds. At each round, items are ranked with respect to $Pref$. Then, the learner receives feedback from the environment. The feedback is assumed to be an arbitrary set of rules of the form "i_1 should be preferred to i_2". Given that $Pref$ is a linear combination of n primitive functions, i.e., $Pref(i_1, i_2) = \sum_{i=j}^{n} w_j F_j(i_1, i_2)$, at each round the weights w_j are updated with regards to the *loss* of a function F with respect to the user feedback, where *loss* is the normalized sum of disagreements between function and feedback.

Moreover, applying machine learning techniques for learning ranking functions has recently attracted much attention in the research literature (e.g., [Burges et al. 2005; Zhai and Lafferty 2006; Zha et al. 2006]).

5.1.2 *Preference Revising.* Preferences change, so one should not base a practical approach on a theory that presupposes fixed preferences. Preferences can change even during examination of alternatives, i.e., during searching. For example, *negotiation* offers a setting that shows the ease with which preferences can change over time and the use of preference expressions to provide succinct instructions for controlling behavior [Doyle 2004]. Of course, one can sometimes prescribe patterns

of change in advance by identifying specific dependencies on particular aspects of the context in which decisions will be made, but such foreknowledge is not always available. Recently, there are approaches focusing in preference change. Freund [2004] studies the problem of revising rational preferences. A single revision satisfies the postulates of success and minimal change, while this result can be extended for multiple revisions. Chomicki [2007a] proposes a framework for incremental preference revision. This work considers binary preference relations. A revised preference relation is produced by composing an original preference relation with another preference relation. Three different semantics of preference revision are considered: union, prioritized and Pareto composition. Also, the cases in which the revision fails, i.e., the revised preference relation is cyclic, are identified, while minimality of preference change is guaranteed.

More recently, Mindolin and Chomicki [2008] [2011] focus on preference contraction, that is, the problem of constructing binary preference relations by discarding preferences. Again, the property of minimality of preference change is desirable. Additional properties studied include the preservation of strict partial orders and the protection of preferences by allowing the specification of the preferences that cannot be contracted.

5.1.3 *Other Preference Models and Applications.* Besides preferences for relational data, preferences have also been used in XML query processing. Preference XPATH Kießling et al. [2001] implements the preference model of [Kießling 2002] for XML databases. Amer-Yahia et al. [2007] define two kind of preference rules for personalized XML search: *scoping rules* that are used to expand or restrict the original query result by adding, removing or replacing query predicates (similar to the query expansion approach) and *ordering rules* that specify how to rank answers. Both rules follow the qualitative approach and define partial orders. Cohen and Shiloach [2009] propose a query language for XML that incorporates value and structure preferences. Preference queries are defined as simple exact queries with some optional constraints. Preference queries are also enhanced with *value orderings* that express priorities among contents of variables, using a qualitative approach. Skyline semantics are applied for computing the "best" results of a preference query.

Although preferences have been mainly used for personalizing queries through controlling the order or size of their results, there are many other potential applications that can be investigated further. Qu and Labrinidis [2007] propose a different form of query personalization through the concept of *quality contracts*. Quality contracts combine user preferences for different quality metrics, such as *quality of service* that refers to the query response time and *quality of data* that refers to the freshness of data in the query answer. Besides query processing, another potential application is that of using preferences for managing the content of a cache. Cherniack et al. [2003] introduce a profile language that uses preferences for specifying the relative importance of the data items to be prefetched. Similarly, Kambalakatta et al. [2004] propose a profile-based caching mechanism for mobile environments. Preferences have been also applied to e-business applications. Kießling et al. [2004] and Döring et al. [2005] present an automated electronic sales agent for e-procurement portals. This work includes, among others, a *presentation* component that adapts the presentation of the query results to the user preferences, supporting several human

sales strategies. Finally, Drosou et al. [2009] introduce *preferential subscriptions* to indicate priorities among subscriptions in publish/subscribe systems, so that the matching events are ranked based on the importance of the related subscription.

5.1.4 *Connection with Other Fields.* Although preferences have been traditionally studied in fields such as philosophy [Hansson 2001], psychology [Scherer 2005], decision making [Lichtenstein and P. Slovic 2006] and economics [Fishburn 1999], nowadays computational methods for handling preferences are studied in fields such as artificial intelligence [Wellman and Doyle 1991], human-computer interaction [Linden et al. 1997] and databases. All these fields can lend useful ideas to databases. For example, psychology understands preference nature and impact on human decision making and can help in building more sophisticated database models and mechanisms for handling preferences. Several aspects of decision making can be exploited to enrich database models of preferences, such as the concept of risk.

On the other hand, databases bring some fresh perspectives to the study of preference modeling and handling by introducing computational methods for the elicitation, representation, aggregation and satisfaction of preferences for computational tasks. This broadened scope of preferences leads to new problems for applying preference structures and new kinds of benefits that extend the scope of classic decision support and add new questions, methods and applications to the handling of preferences. Preferences are inherently a multi-disciplinary topic, of interest to AI, Databases, Logic Programming, Operation Research and more.

5.2 Directions for Future Work

We have only begun to understand the variety, nature and representation of preferences, and the appropriate methods for specifying, realizing, effecting, tracking, adapting and revising preferences. Moving forward, we highlight critical open research challenges and directions for future work.

Hybrid preference models: The preference models proposed so far follow either the qualitative or the quantitative approach. While qualitative preferences can express more types of relations than qualitative preferences, with qualitative preference, we cannot distinguish how much better a query answer is compared to another. We could exploit ideas from both philosophies. For instance, we could express preferences, such as thriller movies are preferred over dramas with score 0.7 and dramas are preferred over comedies with score 0.5. Defining such hybrid preference models is challenging. For example, given the preferences above, what is the relationship between thriller and comedy movies? Furthermore, people naturally express their preferences in either way (e.g., “I like comedies a lot” and “I like comedies more than dramas”). We could also define a hybrid preference model that allows expressing both qualitative and quantitative preferences. Then, several interesting issues arise: How can we combine hybrid preferences? How can we rank query results, using both a qualitative or a quantitative approach?

Preferential social search: Usually, in social web systems, users share resources, such as photos, papers, blogs and other personal information. Since such systems become extremely popular and thus, an important component of the web, exploring new ways of searching the social graph is challenging. Building upon previous work, we envision a social system, where users are allowed to restrict

their view of the social graph based on the current context and related preferences. Users will be able to query the social graph and be presented with a diversified sub-graph relevant to their interests. As the time passes, results will be adapted to the new context of the user. For instance, a query about friends' news submitted on Monday morning may begin returning information about various activities of the user's colleagues. By the evening of the same day, the same query may be presenting information about the user's friends and family. From a different perspective, social systems can be used for deducing user preferences. By exploiting the amount of personal information currently available over the social web, interesting knowledge about users can be elicited.

Preferences and diversity: Usually, preferential query processing methods focus on ranking the query results and report only a portion of them, typically the top ranked ones. The top-results are often very similar to each other, since data items containing the same, highly preferable piece of information are all ranked in the top positions, even if they are redundant. Besides pure accuracy achieved by matching the criteria set by the user preferences, there are also other factors that can increase user satisfaction, such as retrieving results on a broader variety of topics, i.e., increasing the diversity of results (e.g., [Vee et al. 2008]). There are two perspectives on achieving diversity: (*i*) avoiding overlap among results, i.e., choosing items that are dissimilar to each other and (*ii*) increasing coverage, i.e., choosing items that cover as many different topics, or preferences, as possible.

Processing preference queries: There are many open problems with regards to both query expansion or personalization and preference operators. In the case of query personalization, the problem of selecting the appropriate preferences for personalizing a query is challenging, since there can be no single best solution. Furthermore, the performance issues of integrating preferences with query processing are still largely unexplored. Whereas there has been a large amount of work for specific preference operators, such as top-*k*, skyline and their variants, there is very little work on integrating more general preference models within the database engine. Finally, note that database systems are typically used by many users. Processing group preferences and integrating them with database query processing is also a very promising topic.

Preferences over uncertain data: Recently, many approaches (e.g., [Benjelloun et al. 2006; Dalvi and Suciu 2007]) focus on modeling and processing uncertain, or probabilistic, data, since such data are common in many applications, such as sensor networks and location tracking. In probabilistic databases, tuples are associated with membership probabilities that define the belief that they should belong to the database. For processing top-*k* queries, both interest scores and probabilities of tuples are taken into account [Soliman et al. 2007; Soliman and Ilyas 2009; Zhang and Chomicki 2009]. Such approaches may act as a springboard for new methods dealing with the problem of expressing general preferences for uncertain data. Designing appropriate preference models and extending processing methods to conform with them are challenging tasks.

REFERENCES

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

Agrawal, R., Rantzau, R., and Terzi, E. 2006. Context-sensitive ranking. In *SIGMOD*. 383–394.

Agrawal, R. and Wimmers, E. L. 2000. A framework for expressing and combining preferences. *SIGMOD 29,* 2, 297–306.

Amer-Yahia, S., Fundulaki, I., and Lakshmanan, L. V. S. 2007. Personalizing XML search in PIMENTO. In *ICDE*. 906–915.

Amer-Yahia, S., Roy, S. B., Chawla, A., Das, G., and Yu, C. 2009. Group recommendation: Semantics and efficiency. *PVLDB 2,* 1, 754–765.

Balke, W.-T., Güntzer, U., and Kiessling, W. 2002. On real-time top-k querying for mobile services. In *CoopIS*. 125–143.

Bartolini, I., Ciaccia, P., and Patella, M. 2008. Efficient sort-based skyline evaluation. *ACM Trans. Database Syst. 33,* 4.

Benjelloun, O., Sarma, A. D., Halevy, A. Y., and Widom, J. 2006. ULDBs: Databases with uncertainty and lineage. In *VLDB*. 953–964.

Borda, J.-C. 1781. *Mémoire sur les élections au scrutin*. Histoire de l'Académie Royale des Sciences.

Börzsönyi, S., Kossmann, D., and Stocker, K. 2001. The skyline operator. In *ICDE*. 421–430.

Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. H., and Poole, D. 2004. CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *J. Artif. Intell. Res. 21,* 135–191.

Boutilier, C., Brafman, R. I., Hoos, H. H., and Poole, D. 1999. Reasoning with conditional ceteris paribus preference statements. In *Proc. of the Sym. on Uncertainty in AI*. 71–80.

Brown, P., Bovey, J., and Chen, X. 1997. Context-aware applications: From the laboratory to the marketplace. *IEEE Personal Communications 4,* 5, 5864.

Bunningen, A. H., Feng, L., and Apers, P. M. G. 2006. A context-aware preference model for database querying in an ambient intelligent environment. In *DEXA*. 33–43.

Burges, C. J. C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., and Hullender, G. N. 2005. Learning to rank using gradient descent. In *ICML*. 89–96.

Chan, C. Y., Jagadish, H. V., Tan, K.-L., Tung, A. K. H., and Zhang, Z. 2006. Finding k-dominant skylines in high dimensional space. In *SIGMOD*. 503–514.

Chen, G. and Kotz, D. 2000. A survey of context-aware mobile computing research. Tech. Rep. TR2000-381, Dartmouth College, Computer Science. November.

Cherniack, M., Galvez, E. F., Franklin, M. J., and Zdonik, S. B. 2003. Profile-driven cache management. In *ICDE*. 645–656.

Chomicki, J. 2002. Querying with intrinsic preferences. In *EDBT*. 34–51.

Chomicki, J. 2003. Preference formulas in relational queries. *ACM Trans. Database Syst. 28,* 4, 427–466.

Chomicki, J. 2007a. Database querying under changing preferences. *Ann. Math. Artif. Intell. 50,* 1-2, 79–109.

Chomicki, J. 2007b. Semantic optimization techniques for preference queries. *Inf. Syst. 32,* 5, 670–684.

Chomicki, J., Godfrey, P., Gryz, J., and Liang, D. 2003. Skyline with presorting. In *ICDE*. 717–719.

Ciaccia, P. 2007. Querying databases with incomplete CP-nets. In *M-Pref*.

Cohen, S. and Shiloach, M. 2009. Flexible XML querying using skyline semantics. In *ICDE*. 553–564.

Cohen, W. W., Schapire, R. E., and Singer, Y. 1999. Learning to order things. *J. Artif. Intell. Res. (JAIR) 10,* 243–270.

Condorcet, J. A. N. 1785. *Éssai sur l' application del' analyse á la probabilitédes décisions rendues á la pluralité des voix*. Kessinger Publishing.

Dalvi, N. N. and Suciu, D. 2007. Efficient query evaluation on probabilistic databases. *VLDB J. 16,* 4, 523–544.

Das, G., Gunopulos, D., Koudas, N., and Tsirogiannis, D. 2006a. Answering top-k queries using views. In *VLDB*. 451–462.

Das, G., Hristidis, V., Kapoor, N., and Sudarshan, S. 2006b. Ordering the attributes of query results. In *SIGMOD*. 395–406.

Delgrande, J. P., Schaub, T., and Tompits, H. 2003. A framework for compiling preferences in logic programs. *TPLP 3,* 2, 129–187.

Dey, A. K. 2001. Understanding and using context. *Personal Ubiquitous Comput. 5,* 1, 4–7.

Döring, S., 0003, S. F., Kießling, W., and Preisinger, T. 2005. Optimizing the catalog search process for e-procurement platforms. In *DEEC*. 39–48.

Doyle, J. 2004. Prospects for preferences. *Computational Intelligence 20*, 2.

Drosou, M., Stefanidis, K., and Pitoura, E. 2009. Preference-aware publish/subscribe delivery with diversity. In *DEBS*. 1–12.

Dwork, C., Kumar, R., Naor, M., and Sivakumar, D. 2001. Rank aggregation methods for the web. In *WWW10*.

Endres, M. and Kießling, W. 2006. Transformation of TCP-net queries into preference database queries. In *M-Pref*.

Endres, M. and Kießling, W. 2008. Optimization of preference queries with multiple constraints. In *PersDB*. 25–32.

Fagin, R. 1996. Combining fuzzy information from multiple systems. In *PODS*. 216–226.

Fagin, R. 1998. Fuzzy queries in multimedia database systems. In *PODS*. 1–10.

Fagin, R. 1999. Combining fuzzy information from multiple systems. *Journal of Computer and System Sciences 58*, 1, 83–99.

Fagin, R., Kumar, R., Mahdian, M., Sivakumar, D., and Vee, E. 2004. Comparing and aggregating rankings with ties. In *PODS*. 47–58.

Fagin, R., Kumar, R., Mahdian, M., Sivakumar, D., and Vee, E. 2006. Comparing partial rankings. *SIAM J. Discrete Math. 20*, 3, 628–648.

Fagin, R., Kumar, R., and Sivakumar, D. 2003. Comparing top-k lists. In *ACM-SIAM symposium on Discrete algorithms*. 28–36.

Fagin, R., Lotem, A., and Naor, M. 2001. Optimal aggregation algorithms for middleware. In *PODS*.

Fagin, R. and Wimmers, E. L. 2000. A formula for incorporating weights into scoring rules. *Theor. Comput. Sci. 239*, 2, 309–338.

Fishburn, P. C. 1999. Preference structures and their numerical representations. *Theoretical Computer Science 217*, 2, 359–383.

Freund, M. 2004. On the revision of preferences and rational inference processes. *Artif. Intell. 152*, 1, 105–137.

Gaasterland, T. and Lobo, J. 1994. Qualified answers that reflect user needs and preferences. In *VLDB*. 309–320.

Georgiadis, P., Kapantaidakis, I., Christophides, V., Nguer, E. M., and Spyratos, N. 2008. Efficient rewriting algorithms for preference queries. In *ICDE*. 1101–1110.

Godfrey, P., Shipley, R., and Gryz, J. 2007. Algorithms and analyses for maximal vector computation. *VLDB J. 16*, 1, 5–28.

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

Güntzer, U., Balke, W.-T., and Kießling, W. 2000. Optimizing multi-feature queries for image databases. In *VLDB*. 419–428.

Güntzer, U., Balke, W.-T., and Kießling, W. 2001. Towards efficient multi-feature queries in heterogeneous environments. In *ITCC*. 622–628.

Hafenrichter, B. and Kießling, W. 2005. Optimization of relational preference queries. In *ADC*. 175–184.

Hansson, S. O. 2001. Preference logic. *Handbook of Philosophical Logic (D. Gabbay, Ed.) 8*.

Holland, S., Ester, M., and Kießling, W. 2003. Preference mining: A novel approach on mining user preferences for personalized applications. In *PKDD*. 204–216.

Holland, S. and Kießling, W. 2004. Situated preferences and preference repositories for personalized database applications. In *ER*. 511–523.

Hristidis, V. and Papakonstantinou, Y. 2004. Algorithms and applications for answering ranked queries using ranked views. *VLDB J. 13*, 1, 49–70.

Ilyas, I. F., Aref, W. G., and Elmagarmid, A. K. 2004. Supporting top-k join queries in relational databases. *VLDB J. 13*, 3, 207–221.

Ilyas, I. F., Beskales, G., and Soliman, M. A. 2008. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv. 40*, 4.

Ilyas, I. F., Shah, R., Aref, W. G., Vitter, J. S., and Elmagarmid, A. K. 2004. Rank-aware query optimization. In *SIGMOD*. 203–214.

Joachims, T. 2002. Optimizing search engines using clickthrough data. In *KDD*. 133–142.

Kambalakatta, R., Kumar, M., and Das, S. K. 2004. Profile based caching to enhance data availability in push/pull mobile environments. In *MobiQuitous*. 74–83.

Kiessling, W. 2002. Foundations of preferences in database systems. In *VLDB*. 311–322.

Kiessling, W. 2005. Preference queries with SV-semantics. In *COMAD*. 15–26.

Kiessling, W., Fischer, S., and Döring, S. 2004. COSIMAB2B - sales automation for e-procurement. In *CEC*. 59–68.

Kiessling, W., Hafenrichter, B., Fischer, S., and Holland, S. 2001. Preference XPATH: A query language for e-commerce. In *Wirtschaftsinformatik*.

Kiessling, W. and Köstler, G. 2002. Preference SQL - design, implementation, experiences. In *VLDB*. 990–1001.

Kossmann, D., Ramsak, F., and Rost, S. 2002. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB*. 275–286.

Koutrika, G. and Ioannidis, Y. 2005a. Constrained optimalities in query personalization. In *SIGMOD*. 73–84.

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

Koutrika, G. and Ioannidis, Y. 2010. Personalizing queries based on networks of composite preferences. *ACM Trans. Database Syst. 35*, 2.

Koutrika, G. and Ioannidis, Y. E. 2004. Personalization of queries in database systems. In *ICDE*. 597–608.

Lacroix, M. and Lavency, P. 1987. Preferences; putting more knowledge into queries. In *VLDB*. 217–225.

Levandoski, J., Mokbel, M. F., and Khalefa, M. E.. 2010. FlexPref: A framework for extensible preference evaluation in database systems. In *ICDE*.

Li, C., Chang, K. C.-C., and Ilyas, I. F. 2006. Supporting ad-hoc ranking aggregates. In *SIGMOD*. 61–72.

Lichtenstein, S. and P. Slovic, T. . T. 2006. New York: Cambridge University Press.

Lin, X., Yuan, Y., Zhang, Q., and Zhang, Y. 2007. Selecting stars: The k most representative skyline operator. In *ICDE*. 86–95.

Linden, G., Hanks, S., and Lesh, N. 1997. Interactive assessment of user preference models: The automated travel assistant. In *International Conference on User Modeling*. 67–78.

Masthoff, J. 2004. Group modeling: Selecting a sequence of television items to suit a group of viewers. *User Modeling and User-Adapted Interaction 14*, 1, 37–85.

Miah, M., Das, G., Hristidis, V., and Mannila, H. 2008. Standing out in a crowd: Selecting attributes for maximum visibility. In *ICDE*. 356–365.

Miele, A., Quintarelli, E., and Tanca, L. 2009. A methodology for preference-based personalization of contextual data. In *EDBT*. 287–298.

Mindolin, D. and Chomicki, J. 2007. Hierarchical CP-networks. In *M-Pref*.

Mindolin, D. and Chomicki, J. 2008. Minimal contraction of preference relations. In *AAAI*. 492–497.

Mindolin, D. and Chomicki, J. 2011. Contracting preference relations for database applications. *AI Journal To Appear*.

Morse, M. D., Patel, J. M., and Jagadish, H. V. 2007. Efficient skyline computation over low-cardinality domains. In *VLDB*. 267–278.

Motro, A. 1988. Vague: A user interface to relational databases that permits vague queries. *ACM Trans. Inf. Syst. 6*, 3, 187–214.

Natsev, A., Chang, Y.-C., Smith, J. R., Li, C.-S., and Vitter, J. S. 2001. Supporting incremental join queries on ranked inputs. In *VLDB*. 281–290.

Nepal, S. and Ramakrishna, M. V. 1999. Query processing issues in image (multimedia) databases. In *ICDE*. 22–29.

Orlovsky, S. A. 1978. Decision making with a fuzzy preference relation. *Fuzzy sets and systems 1*, 155–167.

Ovchinnikov, S. and Roubens, M. 1992. On fuzzy strict preference, indifference, and incomparability relations. *Fuzzy Sets Syst. 49*, 1, 15–20.

Papadias, D., Tao, Y., Fu, G., and Seeger, B. 2003. An optimal and progressive algorithm for skyline queries. In *SIGMOD*. 467–478.

Pei, J., Jin, W., Ester, M., and Tao, Y. 2005. Catching the best views of skyline: A semantic approach based on decisive subspaces. In *VLDB*. 253–264.

Preisinger, T. and Kiessling, W. 2007. The Hexagon algorithm for pareto preference queries. In *M-Pref*.

Qu, H. and Labrinidis, A. 2007. Preference-aware query and update scheduling in web-databases. In *ICDE*. 356–365.

Ross, K. A., Stuckey, P. J., and Marian, A. 2007. Practical preference relations for large data sets. In *ICDE Workshops*. 229–236.

Scherer, K. R. 2005. What are emotions? and how can they be measured? *Social Science Information 44*, 695–729.

Schmidt, A., Aidoo, A. K., Takaluoma, A., Tuomela, U., Laerhoven, K., and de Velde, M. 1999. Advanced interaction in context. In *1st Int'l Symposium on Handheld and Ubiquitous Computing*. 89101.

Soliman, M. A. and Ilyas, I. F. 2009. Ranking with uncertain scores. In *ICDE*. 317–328.

Soliman, M. A., Ilyas, I. F., and Chang, K. C.-C. 2007. Top-k query processing in uncertain databases. In *ICDE*. 896–905.

Stefanidis, K., Drosou, M., and Pitoura, E. 2010. Perk: Personalized keyword search in relational databases through preferences. In *EDBT*. 585–596.

Stefanidis, K. and Pitoura, E. 2008. Fast contextual preference scoring of database tuples. In *EDBT*. 344–355.

Stefanidis, K., Pitoura, E., and Vassiliadis, P. 2006. Modeling and storing context-aware preferences. In *ADBIS*. 124–140.

Stefanidis, K., Pitoura, E., and Vassiliadis, P. 2007a. Adding context to preferences. In *ICDE*. 846–855.

Stefanidis, K., Pitoura, E., and Vassiliadis, P. 2007b. On relaxing contextual preference queries. In *MDM*. 289–293.

Tan, K.-L., Eng, P.-K., and Ooi, B. C. 2001. Efficient progressive skyline computation. In *VLDB*. 301–310.

Tao, Y., Xiao, X., and Pei, J. 2006. Subsky: Efficient computation of skylines in subspaces. In *ICDE*. 65.

Taylor, A. 1995. *Mathematics and politics: Strategy, voting, power and proof*. New York: Springer Verlag.

Torlone, R. and Ciaccia, P. 2002. Finding the best when it's a matter of preference. In *SEBD*. 347–360.

Torlone, R. and Ciaccia, P. 2003. Management of user preferences in data intensive applications. In *SEBD*. 257–268.

Vee, E., Srivastava, U., Shanmugasundaram, J., Bhat, P., and Amer-Yahia, S. 2008. Efficient computation of diverse query results. In *ICDE*. 228–236.

Wellman, M. P. and Doyle, J. 1991. Preferential semantics for goals. In *Proc. of AAAI*. 698703.

Xia, T., Zhang, D., and Tao, Y. 2008. On skylining with flexible dominance relation. In *ICDE*. 1397–1399.

Yi, K., Yu, H., Yang, J., Xia, G., and Chen, Y. 2003. Efficient maintenance of materialized top-k views. In *ICDE*. 189–200.

You, G. and Hwang, S. 2008. Search structures and algorithms for personalized ranking. *Information Sciences 178*, 20, 3925–3942.

Yuan, Y., Lin, X., Liu, Q., Wang, W., Yu, J. X., and Zhang, Q. 2005. Efficient computation of the skyline cube. In *VLDB*. 241–252.

Zadeh, L., Fu, K., Tanaka, K., and Shimura, M. 1975. *Fuzzy sets and their applications to cognitive and decision processes*. Academic Press, New York.

Zha, H., Zheng, Z., Fu, H., and Sun, G. 2006. Incorporating query difference for learning retrieval functions in world wide web search. In *CIKM*. 307–316.

Zhai, C. and Lafferty, J. D. 2006. A risk minimization framework for information retrieval. *Information Processing and Management 42*, 1, 31–55.

Zhang, S., Mamoulis, N., and Cheung, D. W. 2009. Scalable skyline computation using object-based space partitioning. In *SIGMOD*. 483–494.

Zhang, X. and Chomicki, J. 2009. Semantics and evaluation of top-*k* queries in probabilistic databases. *Distributed and Parallel Databases 26*, 1, 67–126.

ZHANG, X. AND CHOMICKI, J. 2011. Preference queries over sets. In *ICDE*.
ZIMMERMANN, H. 1985. *Fuzzy set theory and its applications*. Kluwer Academic, Dordrecht.