Database Preferences – A Unified Model

Roxana Gheorghiu
University of Pittsburgh
roxana@cs.pitt.edu

Alexandros Labrinidis
University of Pittsburgh
labrinid@cs.pitt.edu

Panos K. Chrysanthis
University of Pittsburgh
panos@cs.pitt.edu

ABSTRACT

In this work we present a new model that combines two different types of preferences, qualitative and quantitative. We show how our model can support different types of preferences at different granularity levels and how an application can use these preferences to retrieve a list of tuples. The new model takes advantage of a graph representation of preferences where nodes in the graph are SQL predicates, edges between two nodes describe a qualitative preference and edges from a node to the same node capture quantitative preferences. Each edge is labeled with a numeric value, between -1 and 1, that expresses the intensity of each preference. Using this graph representation, we further show how two different preferences can be combined and applied to the existing query result set to filter the result and identify the most relevant tuples first.

1. INTRODUCTION

Preferences have been studied for many years and they were traditionally applied to philosophy, psychology and economics. In AI they were applied to decision making problems, capturing agents’ goals. With the rapid increase of the Web and personal communication technologies on one hand and the explosion of available data on the other hand, preferences have become a requirement for the scalable processing of large volumes of data. Query personalization techniques have been proposed both by academia and industry, showing a real interest for techniques that can cope with virtually unlimited amounts of data.

In the database domain preferences are seen as soft criteria, used to rank the results in terms of how well they match the query predicate. In contrast, the predicates in the SQL WHERE clause are seen as hard constraints, a non-empty result being returned only when all conditions are met.

The literature classifies the preferences in two types: quantitative and qualitative. Pitoura et al. [6] describe the existing work in this area in terms of preference representation, preference composition, and preference query processing.

Quantitative preferences are described by scores attached to each tuple, for which a preference was expressed. As an example, consider the preference “I like comedies very much”. This can be translated in the following quantitative preference: (“I like comedies”, score = 1). The score denotes users’ interest in one or multiple data tuples. Using these scores we can define a total order over a set of the tuples.

Qualitative preferences are expressed as pairs of tuples. As an example, consider the preference “I like comedies more than dramas”. This can be translated into the following qualitative preference: (“comedies”, preferred over, “dramas”). When put together, these pairs create a partial order among the tuples in the database.

When all database tuples have a score assigned to them they can be ranked from the most preferred to the least preferred. The tuples that do not match any preference can be divided into two categories: equally preferred and incomparable. In the first case, tuples equally preferred can be seen as having the same score assigned, whereas, in the second case, tuples that are incomparable cannot be included in the partial or total order defined by the preferences.

Each type of preference—quantitative and qualitative—has its advantage over the other and there are cases when a user’s preference can be conveniently expressed using one approach but not the other. For example, it is very easy to express a negative preference in the quantitative model by assigning a negative weight to that particular tuple or to a set of tuples that match a given condition. However, there is no easy way to express a negative preference in the qualitative model, since this will require to explicitly list all tuples that are preferred over those that are not preferred.

Kiessling et al. [1] proposed Preference SQL, a framework that can support a hybrid version of both qualitative and quantitative preferences. This can be seen as supporting a local view of preferences because each preference is given by the user at query time. Preference SQL allows the use of a score function to assign a value to an individual tuple which is used only as a quantitative preference and has no meaning for the binary relation that describes a qualitative preference. We will show in Sec. 4 how this lack of scoring of a binary relation, describing qualitative preference, can lead to an unexpected ordering of query results.

In this paper we propose a unified model which is based on graphs and which integrates qualitative and quantitative preferences by means of preference intensity values (scores) and user profiles. Our approach benefits from creating a global view of preferences stored in user profiles that can be used to correctly rank the query results, from the most
preferred to the least preferred tuple. In our model the system, rather than the user, determines which preferences are applicable to her specific query.

The formal underpinning of our unified model is a preference graph. Each node in the graph represents a query condition; a directed link between two nodes is used to define a preferred order between tuples. We express quantitative preferences using self addressed edges (i.e., have the same starting and ending point). Qualitative preferences are represented by edges between two different nodes, and each edge is labeled with a value that represents the intensity of the preference.

The system stores the preferences in a database of user profiles, each preference graph as an adjacency matrix. Each user maintains her own qualitative and quantitative preferences over the database tuples or attributes along with their intensity values. Although not discussed in this paper, a user can define multiple profiles which can be used in different contexts [5]. Also not discussed in this paper is a specific query language of preferences. The goal of our query model is to identify the most relevant tuples first (i.e., Top-K ones).

**Contributions:** The contributions of this paper are as follows:

- We propose a new model that incorporates qualitative and quantitative preferences expressed as a preference graph. Our model can be used to define both tuple-based and predicate-based preferences and supports preferences at different levels of granularity (i.e., tuple level vs. attribute level).
- We show how the intensity of preference values in the preference graphs, stored in a user profile, are used by the system to define a user-specific order over the tuples in the database.

**Roadmap:** Sec. 2 presents our unified model for preferences, its implementation and preference specifications through examples. Sec. 3 introduces the composition techniques used to combine the two preference types. Sec. 4 covers the related work and the differences between our proposed model and existing ones and Sec. 5 concludes with future work.

## 2. Unified Model for Preferences

A graph representation is the most natural way of exemplifying the connections between tuples in a database and visually depicting their relationships. The purpose of our preference graph is to connect two different preference approaches into a unified model.

**Definition 1.** We define the graph of preferences \(PG = (PV, PE)\) as a labeled directed graph where:

- \(PV\) is the set of vertices; each vertex represents a tuple in the database or a query predicate (e.g., a set of tuples).
- \(PE\) is the set of edges; each edge \((v_i, v_j, s)\) defines a direction and is labeled with a score \(s\). An edge from \(v_i\) to \(v_j\) captures a qualitative preference (e.g., the value in vertex \(v_i\) is preferred over the value in vertex \(v_j\)) whereas an edge from \(v_i\) to itself will describe a quantitative preference. The score \(s\) is a value between -1 and 1 that captures the preference intensity.

Using the preference graph description above, one preference is defined by the triplet \((v_i, v_j, s)\), where \(v_i, v_j \in PV\) and \((v_i, v_j, s) \in PE\). When \(i=j\), the triplet \((v_i, v_j, s)\) represents a qualitative preference, and when \(i \neq j\), it represents a quantitative preference.

In our preference graph, intensity is a value between -1 and 1. All negative values are used to express negative preferences, -1 being used to express complete dislike. In a similar way, all positive values are used to express positive preferences and 1 is used to capture the most preferred tuple. Zero is a special value used to express equally preferred tuples, in the case of qualitative preferences, and indifference, in the case of quantitative preferences.

For a quantitative preference, the intensity value expresses the preference strength for one particular tuple (or set of tuples) over all other tuples in the database. In this case, intensity has the semantics of the score, and a large intensity value describes a strong preference towards that particular tuple (or set of tuples).

For a qualitative preference, the intensity value expresses the preference strength for one tuple (or set of tuples) over another tuple (or set of tuples). In this case, a small positive value will express a similarity on preferences (i.e., one tuple is almost as preferred as the other tuple).

Intensity can be a constant value or a function to allow dynamic ranking of preferences. As an example, consider the preference: “I like recent comedies”, where recent can be expressed as a function on the year a movie was released and normalized in the proper range (i.e., \([-1, 1]\)).

A vertex in the graph can represent a single tuple in the database (tuple preference graph), or a set of tuples if it is defined as a query predicate (predicate preference graph).

A tuple-based preference graph is usually not scalable because, for each tuple that matches a preference, a new vertex is created in the preference graph. However, this type of preference graph can be seen as a materialized database view and is useful especially in cases when the preference has a low probability of changing.

A predicate-based preference graph is a scalable version of the tuple-based one, since a vertex matches multiple tuples, and it is used for preferences that apply to a large set of tuples. This type of preference graph is also useful for preferences that are removed and reinserted often because it is much easier and efficient to add/remove only one vertex in the graph rather than all vertices that represent tuples that match a particular preference.

From the representation point of view, both types of graphs are similar, and for this reason we will make a detailed presentation of a tuple-based preference graph in Sec. 2.1. The predicate-based preference graph model follows the same specifications and, given the space constraints, we will only point out the differences, in Sec. 2.2.

In our model, preference inconsistencies or conflicts occur either (1) between two qualitative preferences of opposite direction, or (2) when a qualitative preference connects a weak quantitative preferences to a strong one with a positive intensity value. The former case is manifested in the form of cycles with the exception of cycles involving a single node. Currently, we prevent inconsistencies by assuming directed acyclic graphs and not allowing qualitative preferences if they conflict with existing quantitative ones. As part of our future work, we will investigate more sophisticated ways to deal with inconsistencies beyond rejecting preferences.

In our model, user preferences are stored in user profiles and a preference graph is stored using an adjacency matrix.
A cell in an adjacency matrix contains the intensity value associated with one particular preference when a preference is defined, and is empty otherwise. Following this definition, the intensity values of all quantitative preferences are values on the diagonal of adjacency matrix whereas the intensity of all qualitative preferences are values in all other cells.

2.1 Preference Specification Examples

In the next four subsections we exemplify how key types of preferences can be expressed using our proposed preference graph model and how intensity values are used to give an order over the tuples in the database.

Definition 2. Let PrefSet be a generic set of preferences over a relational database. We consider any correctly expressed SQL predicate that can match one or more tuples as a preference. We define function Preferred(), with two arguments, as: Preferred: PrefSet × PrefSet → [-1,1] with the following properties:

- If function’s arguments coincide, the function describes a quantitative preference.
- If function’s arguments differ, the function describes a qualitative preference.
- The value of the function represents the intensity of that particular preference.

In our scenarios, we will use the classic toy example of Movies table (see Table 1). We start with an empty preference graph and we incrementally add new preferences in this graph. In Figures 1-5, along with the graph representation, we also display the adjacency matrix that results when a new preference is added and will be used to internally store the preference graph. For simplicity, we will make a detailed presentation for the tuple-based preference case, but, as we mentioned in the previous section, the predicate-based preference case works similarly.

2.1.1 Negative Preference (Fig. 1)

Assume a free text representation of preference, as follows:

Free text representation: “I don’t like horror movies. Intensity=-1”.

This type of preference is very useful in cases where the user knows what she does not want to see in the final query result. It can be easily expressed using a qualitative preference (i.e., by assigning an intensity value equal to -1) but is virtually impossible using a quantitative preference approach.

If we would rewrite this in logic representation, making use of the Preferred() function, we get:

Logic representation: ∀m ∈ Movie: m[genre] = “horror” → Preferred(m, m) = -1

In our database example tuple m2 will match this preference therefore it will be incorporated in the preference graph. Following our proposed graph-based model we have a graph representation:

Graph representation: We create one node, m2, with a self addressed edge labeled -1. Since the preference graph is empty, this will be the first node added to the graph.

2.1.2 Relative Preference (Fig. 2)

Free text representation: “If two movies have the same genre, I prefer the longer movie with intensity 0.8”.

This is an example of a qualitative preference that cannot be expressed as a quantitative preference. This preference constructs a partial order between two tuples that match one common condition (i.e., they have the same genre) but are different in another (i.e., different duration). As an example from our database, tuple m4 will be preferred over tuple m5, and tuple m3 will be preferred over tuple m1.

Logic representation: ∀m1, m2 ∈ Movie: m1[genre] = m2[genre] ∧ m1[duration]>m2[duration] → Preferred(m1, m2) = 0.8

Graph representation: Four new nodes will be added to the graph: m1, m3, m4, and m5. Also, two edges, from m3 to m1 and from m4 to m5, labeled 0.8 will capture the qualitative aspect of the preference. The adjacency matrix will be also changed accordingly.

2.1.3 Intensity for Qualitative Preference (Fig. 3)

A qualitative preference alone is defined only in terms of pairs of tuples (usually the first tuple in the pair is preferred over the second tuple), but it cannot capture how strong the feeling related to that particular preference is. Our model incorporates an intensity value to cope with this problem. The next example will illustrate this situation.

Free text representation: “I like drama movies a bit more than horror movies with intensity 0.2”.

In our database, tuples m3 and m1 are preferred over m2.

Logic representation: ∀m1, m2 ∈ Movie: m1[genre] = “drama” ∧ m2[genre] = “horror” → Preferred(m1, m2) = 0.2

Graph representation: An edge, labeled 0.2, will be added between vertices m3 and m2, and between m1 and m2.

Figure 1: Specification of negative preferences (a) Graph; (b) Associated adjacency matrix

Figure 2: Specifications of relative preferences (a) Graph; (b) Associated adjacency matrix

Figure 3: Specification of Intensity of Preferences (a) Graph; (b) Associated adjacency matrix
<table>
<thead>
<tr>
<th>movie_id</th>
<th>title</th>
<th>year</th>
<th>director</th>
<th>genre</th>
<th>language</th>
<th>duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1</td>
<td>Casablanca</td>
<td>1942</td>
<td>M. Curtiz</td>
<td>drama</td>
<td>english</td>
<td>102</td>
</tr>
<tr>
<td>m2</td>
<td>Psycho</td>
<td>1960</td>
<td>A. Hitchcock</td>
<td>horror</td>
<td>english</td>
<td>109</td>
</tr>
<tr>
<td>m3</td>
<td>Schindler’s List</td>
<td>1993</td>
<td>S. Spielberg</td>
<td>drama</td>
<td>english</td>
<td>195</td>
</tr>
<tr>
<td>m4</td>
<td>White Christmas</td>
<td>1954</td>
<td>M. Curtiz</td>
<td>comedy</td>
<td>english</td>
<td>120</td>
</tr>
<tr>
<td>m5</td>
<td>The Adventures of Tintin</td>
<td>2011</td>
<td>S. Spielberg</td>
<td>comedy</td>
<td>english</td>
<td>110</td>
</tr>
</tbody>
</table>

Table 1: The Movie Relation

Figure 4: Specification of set preferences (a) Graph; (b) Associated adjacency matrix

2.1.4 Sets (Fig. 4)
Preference over a set of tuples is another example that can be expressed as a qualitative preference but not as a quantitative one. For example, assume the following preference:

Free text representation: “From a collection of movies D, I prefer one comedy and for as many movies as possible to have the same director. Preference intensity is 0.3”.

Logic representation:∀m1, m2 ∈ Movie, m1 ≠ m2; m1[genre]=“comedy” ∧ m2[director]=m1[director]→ Preferred(m1, m2)=1 ∧ Preferred(m2, m1)=0.3

In our database, tuples m4 and m5 are preferred because they refer to comedies and the pairs of tuples: (m5, m3) and (m4, m1) are almost similar in preference (i.e., intensity value is 0.3) because they have the same director.

Graph representation: If two movies have the same director then an edge is created between the two tuples and a small value is used for intensity (e.g., 0.3) to suggest that the two tuples are similar in terms of preference. Tuple m4 is slightly more preferred than tuple m3 because it is also a comedy. For any tuple that represents a comedy, a self-addressed edge is created with a preference score (i.e., label) of 1. Moreover, there are two tuples in our database that represent a comedy: m5 and m4. To emphasize that comedies are equally preferred in this query, one edge (from m5 to m4) will be created in our graph and labeled with intensity 0. Notice that another edge, from m4 to m5 can be added in the graph but since these edges are labeled with 0, and there is already one edge from m4 to m5 we keep only the one that does not overlap.

2.2 Predicate preference
Sec. 2.1 described in detail the idea of our proposed preference graph model for the case in which each node in the graph is associated with a tuple in the database. For the case of a predicate-based preference graph, everything mentioned in the previous section holds. The only difference in this case is that each row in the adjacency matrix (and each vertex in the graph) is a predicate which possibly matches multiple tuples in the database. As an example, consider the following two preferences:

P1: “I prefer comedy movies over drama movies”, score = 0.7 which can be translated as: P1_1 is preferred over P1_2 with intensity 0.7, where the new preferences are defined as P1_1: genre=“comedy” & P1_2: genre=“drama”

P2: “I prefer drama movies with intensity=0.6”, which can be translated into: genre=“drama”, score =0.6

Given that P1_2 is the same as P2, the adjacency matrix for these two predicate preferences is shown in Figure 5.

Figure 5: A predicate-based preference graph and its adjacency matrix

2.3 Preferences at different granularity
Our proposed preference model can be used at different granularity levels. In previous sections we showed how it can be used to handle preferences that can be defined in one model but not in the other. In this section we describe how we can use this same model to support preferences over attributes, in addition to preferences over values. A new preference graph over attributes and its corresponding adjacency matrix is created with vertices that contain attributes instead of tuples.

Free text representation: “I am interested in directors but not in genres. Preference intensity is 0.8”.

Graph Representation (Fig. 6): Two nodes are added into the attribute preference graph: Director attribute (A1) and Genre attribute (A2). A2 has a self-addressed edge labeled with -1 that captures a negative preference towards Genre of the movie. We also create an edge from A1 to A2 and label it with 0.8 to express the user is more interested in directors than genre.

This preference example states that attribute director is more important than attribute genre. For the cases where attribute values are missing, the tuples that have a value for the director field but no value for the genre field will be preferred over tuples without a value for director but with a value for genre. Another useful case for this type of preference will be when it is combined with preferences...
over attribute values. For example, assume the following two extra preferences: “I prefer comedy movies” and “I prefer movies directed by Spielberg”. In this case, tuple m5 will be preferred over tuple m4 because the director is more important than the genre of the movie.

3. PREFERENCE COMPOSITION

Each preference in a set of preferences can be applied to one or more tuples in the database. When two preferences affect the same tuple(s) it is necessary to have a mechanism of combining them into a single preference.

The literature describes two types of qualitative composition methods based on attitude: overriding attitude and combinatorial attitude [6]. In the overriding attitude one preference has priority over the other, meaning that the lower priority preference is applicable only if the higher priority preference is not. In the combinatorial attitude, as the name suggests, both preferences are used and combined. All techniques described in the literature handle the composition for the same type of preferences (i.e., combining two quantitative preferences or combining two qualitative preferences). Our model is designed to handle also compositions of one qualitative and one quantitative preference.

Recall that, in our unified model there is a score (i.e., the intensity) associated with each preference. In the case of a quantitative preference, the score is assigned to each tuple that matches the preference. In the case of qualitative preference, the score is attached to a pair of tuples.

By using intensity values to create an order over the preferences. This order will be further used during the query time to retrieve a list of tuples arranged from the most preferred to the least preferred. In our preference graph model the value on a self addressed edge represents the preference intensity towards all the tuples that match that particular predicate. If this value is not provided by the user, the system will not be able to rank the tuples that match the predicate in the final result list. For this reason the system always assigns a default value, equal to 0.5, when one is not provided by the user. However, if the new vertex is introduced by a qualitative preference with intensity value that connects it to another vertex as P2_1 in Fig. 7, the system replaces the default value for P2_1 with one computed based on the values on the two existing edges. The system always consider only the intensity value on the edge that connects the new vertex with the exiting one and the intensity value on the vertex the new node connects to.

Consider the example preferences captured by the graph in Fig. 7:

- P1: “I prefer red cars, score = 0.6 (quantitative preference)
- P2: “I prefer blue cars over red cars, score = 0.9 (qualitative preference)

With an existing vertex in the graph, P1, we introduce P2. For P2 we first split it into two basic preferences, P2_1: “I prefer blue cars”, P2_2: “I prefer red cars”. Since P2_2 is the same as P1, only P2_1 needs to be added to the preference graph. The vertex P1 has a self addressed edge labeled 0.6, and a new edge, labeled 0.9, will connect P2_1 to P1.

Given the new qualitative preference edge, the system in this case computes the value v for the new vertex P2_1, using the function described in Def. 3. As a result, the intensity value computed is greater than the value in the vertex that it connects to (i.e., P1) and for this example is equal to 0.75.

PREFERENCE COMPOSITION

Each preference in a set of preferences can be applied to one or more tuples in the database. When two preferences affect the same tuple(s) it is necessary to have a mechanism of combining them into a single preference.

The literature describes two types of qualitative composition methods based on attitude: overriding attitude and combinatorial attitude [6]. In the overriding attitude one preference has priority over the other, meaning that the lower priority preference is applicable only if the higher priority preference is not. In the combinatorial attitude, as the name suggests, both preferences are used and combined. All techniques described in the literature handle the composition for the same type of preferences (i.e., combining two quantitative preferences or combining two qualitative preferences).

Our model is designed to handle also compositions of one qualitative and one quantitative preference.

Recall that, in our unified model there is a score (i.e., the intensity) associated with each preference. In the case of a quantitative preference, the score is assigned to each tuple that matches the preference. In the case of qualitative preference, the score is attached to a pair of tuples.

We use intensity values to create an order over the preferences. This order will be further used during the query time to retrieve a list of tuples arranged from the most preferred to the least preferred. In our preference graph model the value on a self addressed edge represents the preference intensity towards all the tuples that match that particular predicate. If this value is not provided by the user, the system will not be able to rank the tuples that match the predicate in the final result list. For this reason the system always assigns a default value, equal to 0.5, when one is not provided by the user. However, if the new vertex is introduced by a qualitative preference with intensity value that connects it to another vertex as P2_1 in Fig. 7, the system replaces the default value for P2_1 with one computed based on the values on the two existing edges. The system always consider only the intensity value on the edge that connects the new vertex with the exiting one and the intensity value on the vertex the new node connects to.

Consider the example preferences captured by the graph in Fig. 7:

- P1: “I prefer red cars, score = 0.6 (quantitative preference)
- P2: “I prefer blue cars over red cars, score = 0.9 (qualitative preference)

With an existing vertex in the graph, P1, we introduce P2. For P2 we first split it into two basic preferences, P2_1: “I prefer blue cars”, P2_2: “I prefer red cars”. Since P2_2 is the same as P1, only P2_1 needs to be added to the preference graph. The vertex P1 has a self addressed edge labeled 0.6, and a new edge, labeled 0.9, will connect P2_1 to P1.

Given the new qualitative preference edge, the system in this case computes the value v for the new vertex P2_1, using the function described in Def. 3. As a result, the intensity value computed is greater than the value in the vertex that it connects to (i.e., P1) and for this example is equal to 0.75.

Our model is designed to handle also compositions of one qualitative and one quantitative preference.

Recall that, in our unified model there is a score (i.e., the intensity) associated with each preference. In the case of a quantitative preference, the score is assigned to each tuple that matches the preference. In the case of qualitative preference, the score is attached to a pair of tuples.

We use intensity values to create an order over the preferences. This order will be further used during the query time to retrieve a list of tuples arranged from the most preferred to the least preferred. In our preference graph model the value on a self addressed edge represents the preference intensity towards all the tuples that match that particular predicate. If this value is not provided by the user, the system will not be able to rank the tuples that match the predicate in the final result list. For this reason the system always assigns a default value, equal to 0.5, when one is not provided by the user. However, if the new vertex is introduced by a qualitative preference with intensity value that connects it to another vertex as P2_1 in Fig. 7, the system replaces the default value for P2_1 with one computed based on the values on the two existing edges. The system always consider only the intensity value on the edge that connects the new vertex with the exiting one and the intensity value on the vertex the new node connects to.

Consider the example preferences captured by the graph in Fig. 7:

- P1: “I prefer red cars, score = 0.6 (quantitative preference)
- P2: “I prefer blue cars over red cars, score = 0.9 (qualitative preference)

With an existing vertex in the graph, P1, we introduce P2. For P2 we first split it into two basic preferences, P2_1: “I prefer blue cars”, P2_2: “I prefer red cars”. Since P2_2 is the same as P1, only P2_1 needs to be added to the preference graph. The vertex P1 has a self addressed edge labeled 0.6, and a new edge, labeled 0.9, will connect P2_1 to P1.

Given the new qualitative preference edge, the system in this case computes the value v for the new vertex P2_1, using the function described in Def. 3. As a result, the intensity value computed is greater than the value in the vertex that it connects to (i.e., P1) and for this example is equal to 0.75.
4. RELATED WORK

Many solutions have been proposed for working with preferences [6]. In most of the cases the designed systems can handle only one type of preference (e.g., qualitative or quantitative). Our proposed model combines these two different approaches into a unified model. To our best knowledge, there is no prior work that handles qualitative preferences in conjunction with quantitative preferences with the exception of Preference SQL [1] as mentioned in the Introduction.

Preference SQL system introduces a new clause, PREFERRING, in which the user can state their preferences relative to the current query. All preferences are connected with an AND operator except for the case when a qualitative preference is defined, in which case an ELSE operator is used to suggest that if the first criteria is not met, then the second one should be used. Also, in order to put a priority on preferences over different attributes, a PRIOR TO operator is provided. In this framework users need to fully describe their preferences for each query, in contrast to our approach, in which preferences are stored in user profiles and the system decides their applicability for each query. Because of this difference we refer to Preference SQL as a local approach and ours as a global one. To illustrate the difference between these approaches, consider the following example.

Assume the two preferences: “I like white cars slightly better than yellow cars” and “I prefer cars between 4 and 6 years old”. Also assume that we have the following tuples in the database: t1: (color=yellow, age=4), t2: (color=white, age=20) and t3: (color=red, age=5). In Preference SQL we have two ways to write this preference, in the PREFERRING clause. First way is to state that both preferences are equally important, which results in the following preferring clause: PREFERRING color IN (‘white’) ELSE (‘yellow’) AND age BETWEEN 4 AND 6. The query returns again tuple {t2} as the most preferred.

Another way is to consider one preference more important than the other. For example, if we consider color more important than age then the preferring clause has the following form: PREFERRING color IN (‘white’) ELSE (‘yellow’) PRIOR TO age BETWEEN 4 AND 6. In this case the query returns again tuple {t2} as the most preferred.

However when submitting this query, we expect {t1} to be the most preferred tuple because both preferences can be applied on tuple t1 whereas tuple t2 is preferred only in the first preference and tuple t3 is preferred only in the second preference. As explained in the previous sections, the meaning of intensity score in a qualitative preference is to express the strength of the relationship between two basic preferences, in this case color=“white” and color=“yellow”. The user specified that color=“white” is slightly better than color=“yellow” which makes two tuples with these values for attribute color to be very close in terms of preferences. In contrast, Preference SQL can attach a score to each tuple but in this case the score will refer to that particular tuple in respect to all other tuples in the database and, therefore, will not be able to capture the connection between two tuples as a qualitative preference does. Our system can correctly rank tuple t1 as the first tuple as seen in Sec. 3.

The work done by Koutrika and Ioannidis [3] is the other most related to ours. In their work, the preferences are kept as query predicates with an intensity value attached. In contrast to our work, they only record quantitative preferences and they use them to create a preference network (i.e., a directed acyclic graph) that will allow an efficient identification of relevant preferences. This graph is used to depict the relation between preferences (i.e., each node in the network refers to a subclass of entities that its parent refers to) whereas in our case the graph depicts the flow of the preferences from the most preferred ones to the least preferred.

In contrast to the work of Koutrika and Ioannidis [3], our work keeps track of preferences in any form (qualitative and quantitative) and our graph representation captures user specific order of tuples as they will show up in the final response after preferences are applied.

5. CONCLUSIONS AND FUTURE WORK

In this work we presented a new model that combines two different types of preferences often studied before, qualitative and quantitative. We showed how our model could support different types of preferences at different granularity levels and how an application can use this preference model to retrieve a list sorted from the most preferred to the least preferred tuples. This is an important desideratum that accommodates two powerful approaches of preferences using a unified model.

We are currently implementing this model in Astroshelf [4]. Preferences are added using a simple web form where users can define one or more preferences, using a single predicate or a disjunction of conjunctions when multiple predicates are part of the same preference. As part of our future work, we are implementing the full version of the theoretical model described here to find its strengths and weaknesses, in particular in terms of its performance/scalability. Another interesting problem we plan to investigate is the recommendation of interesting data tuples, by employing collaborative filtering techniques over the set of user-specified preferences.

6. ACKNOWLEDGMENTS

This research was supported in part by NSF career awards IIS-0746696 and IIS-0952720 and NSF grant OIA-1028162.

7. REFERENCES