A Database Approach for Categorical Preferences on Hierarchies

Patrick Roocks and **Florian Wenzel** and **Olena Rudenko** and
Markus Endres and **Werner Kießling**
University of Augsburg
86135 Augsburg, Germany
firstname.lastname@informatik.uni-augsburg.de

Abstract

In the last decade there has been much interest in preference query processing for various applications like personalized information or decision making systems. Many attributes from the queried datasets are typically organized in hierarchies of tree-shaped structure. In database systems, this fact is commonly represented by a connection of each tuple to its parent node. SQL dialects provide various constructs for processing this kind of data. We introduce a novel technique to handle hierarchies in preference queries by extension of preference algebra. For this purpose we define new preference constructors to specify wishes for attribute values associated with hierarchies. Our approach allows the intuitive definition of a transitive closure in database systems and overcomes the massive recomputation required by standard methods.

1 Introduction

Data occurring in information systems and handled by database management systems (DBMS) can be differentiated by two fundamentally different types: 1) Continuous data such as prices or distances and 2) Discrete data such as quantities, categories, or job positions. Many attributes of the latter type are typically organized in hierarchies, e.g., ontologies or organization charts of companies. Every instance of such an attribute is usually associated with a node or leave of a tree-shaped structure. In DBMS, this is commonly represented with a dataset in which every tuple is connected to its parent node. SQL dialects of popular DBMS offer a variety of constructs for querying tree-like structures, e.g., **`CONNECT BY`** in Oracle[1] or **`WITH RECURSIVE`** in PostgreSQL[2].

Beyond this, preferences in DBMS – as shown by a recent survey [Stefanidis *et al.*, 2011] – as well as preferences in artificial intelligence and computational social choice theory [Rossi *et al.*, 2011; Kaci, 2011] are a well established framework to create personalized information systems. Moreover, preference database queries become more and more important in decision support environments [Golfarelli and Rizzi, 2009a] as an effective method to reduce very large datasets to a small set of highly interesting results. With the commercial EXASolution system [Mandl *et al.*, 2015] there exists a scalable implementation in a start-of-the-art DBMS. The Preference SQL system [Kießling *et al.*, 2011] and the rPref package (formerly R-Pref [Roocks and Kießling, 2013]) for the statistical computing language R are research prototypes to develop and evaluate new preferences for that system.

Preferences on continuous domains are typically Skyline selections as introduced in [Börzsönyi *et al.*, 2001]. The idea is to find Pareto optimal tuples, where different dimensions are optimized simultaneously. For example the following Preference SQL query

```
SELECT *
FROM R
PREFERRING a LOWEST AND b HIGHEST;
```

returns all tuples from the dataset R for which no dominating tuples (lower in attribute a and higher in attribute b) exist.

For preferences on discrete domains a variety of constructors is defined in [Kießling, 2002] and implemented in [Kießling *et al.*, 2011]. In the present work we want to bridge the gap between a *natural preference query language* (i.e., similar to natural language, following the idea of SQL) and tree-like structures. For example, we are interested in all objects belonging to a given category x or some sub-category. Technically this can be (naively) done formulating

```
SELECT *
FROM R
PREFERRING cat_id IN [subquery(x)];
```

where subquery(x) computes the transitive closure of category x. However, this leads to lengthy and queries which are far away from a "natural language". Moreover, this causes a massive recomputation of the transitive closure if there are many of these subqueries, where the different values of `x` are children of each other w.r.t. the category hierarchy.

To overcome these drawbacks, we present novel preference constructors for hierarchical categorical attributes that provide an intuitive expression of preferences and at the same time optimize evaluation by eliminating recomputations. The core of this approach is highlighted by the following example operating on data of an online outdoor platform.

[1] `http://www.oracle.com`
[2] `http://www.postgresql.org`

Example 1. *Let* `tour` *be the dataset of hiking tours as given in Table 1. Each hiking tour has a* name*, a* difficulty *(easy, medium, difficult) and a unique identifier* id*.*

Table 1: Sample dataset of outdoor activities.

tour	id	cat_id	difficulty	name
	1	1	medium	By bike around the lake
	2	2	easy	eBike test track
	3	4	medium	By foot to the top
	4	5	difficult	Via ferrata – not for beginners
	5	6	easy	Walk to the abbey
	6	7	easy	Way of St. James

The cat_id *corresponds to a category which is listed in the* hierarchy table*, cp. Table 2. For example, the outdoor activity "By bike around the lake" has category 1, which is a "CycleTour". For a better understanding we present this hierarchy in a tree structure in Figure 1.*

Table 2: Sample dataset of categories for outdoor activities.

category	id	parent_id	name
	0	NULL	OutdoorActivity
	1	0	CycleTour
	2	1	eBikeTour
	3	1	LongCycleTour
	4	0	HikingTour
	5	4	ViaFerrata
	6	4	PilgrimPath
	7	6	LongPilgrimPath

Now, consider our new construct for preference handling on hierarchies.

```
SELECT id
FROM tours
PREFERRING cat_id SUCCEEDING 1 AND
           cat_id PRECEDE_NEAR 7;
```

The keyword **PREFERRING** *initiates the preference part of the query,* **AND** *expresses a Pareto composition stating the equal importance of two preferences. For now we assume that the Preference interpreter knows that the hierarchy according to the attribute* `cat_id` *is stored in the* `category` *table, given in Table 2. Later on we will describe this in detail.*

The following base preference constructors of aforementioned query are novel contributions of this work:

- `cat_id` **SUCCEEDING** `1`*: The user prefers all activities which have category id* 1 *or one of the sub-categories, i.e., those with an id in* $\{2, 3\}$*. Given Figure 1, a preference for cycle tours of all kinds is expressed.*

- `cat_id` **PRECEDE_NEAR** `7`*: Those activities with category id* 7 *are preferred. If this not feasible the activities with the parent category id* 6 *and its children are preferred. If there are also no results available, the next parental node in the category is considered, and so on. Given Figure 1, long pilgrim paths would be the first category of choice. Next are pilgrim paths. If none of those categories is applicable, hiking tours and via ferrata are the next best alternatives. Finally, outdoor activities in general as well as the left sub-tree are preferred.*

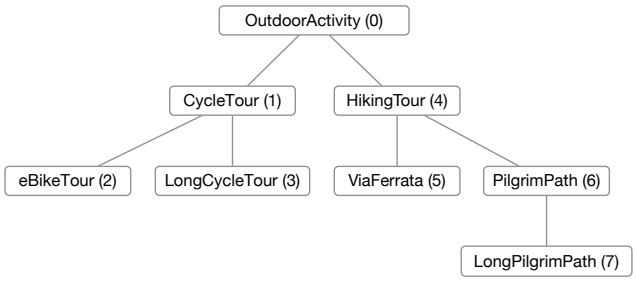

Figure 1: Hierarchy of activities of an outdoor platform

As the activities $\{1, 2\}$ *(contained in categories* $\{1, 2\}$*) are optimal w.r.t. the first preference (***SUCCEEDING***) and the "Way of St. James" with id* 6 *(contained in category* 7*) is optimal w.r.t. the second preference (***PRECEDE_NEAR***), the result of this query is* $\{1, 2, 6\}$*.*

In this paper we introduce two new preference constructors to specify wishes for attribute values associated with hierarchies. Based on this, we extend the syntax of the Preference SQL query language. We illustrate our approach with the help of use cases in the domain of recommender system for outdoor activities. To evaluate our approach, we implemented a prototype based on *rPref* which can be obtained from [Roocks, 2015].

The remainder of this paper is organized as follows: Section 2 recapitulates essential concepts of the used preference model. Section 3 introduces our novel handling of hierarchies in preference algebra and extends the syntax of the Preference SQL query language. Afterwards, we present use cases derived from an ongoing industry project in Section 4. Section 5 highlights related work. Finally, we conclude in Section 6.

2 Preferences in Database Systems

Preference queries in database systems have been in focus for some time, leading to diverse approaches, e.g., [Chomicki, 2003; Kießling, 2002; Kießling *et al.*, 2011]. We follow the preference model from [Kießling, 2005], where a preference $P = (A, <_P)$ is a strict partial order on the domain of A. Thus $<_P$ is irreflexive and transitive. The term $\mathbf{x} <_\mathbf{P} \mathbf{y}$ is interpreted as "*I like y more than x*". Two tuples x and y are *indifferent*, if $\neg(x <_P y) \wedge \neg(y <_P x)$, i.e., neither x is better than y nor y is better than x.

The *optimal (or maximal) objects* of a preference $P = (A, <_P)$ on an input database relation R contain all tuples that are not dominated by any other tuple R w.r.t. the preference. These objects are computed by the preference selection operator $\sigma[P](R)$ (called *winnow* by [Chomicki, 2003]) that finds all best matching tuples t for P, where $t.A$ is the projection to the attribute set A.

$$\sigma[P](R) := \{t \in R \mid \neg\exists t' \in R : t.A <_P t'.A\} \quad (1)$$

The retrieval of best-matching results hence follows a Best-Matches-Only (BMO) query model that retrieves exact matches if such objects exist and best alternatives else.

To specify a preference, the framework follows an inductive approach of base preference constructors that can be combined via complex preference constructors. Subsequently, we present some selected constructors which are used in this paper. A detailed overview can be found in [Kießling, 2002; Kießling, 2005; Kießling *et al.*, 2011]. The new preference constructors for categorical preferences on hierarchies will be introduced in the next section.

2.1 Base Preference Constructors

Preferences on single attributes are called *base preferences*. There are base preference constructors for *continuous* (*numerical*), *discrete* (*categorical*), *temporal*, and *spatial* domains. Most of them can be expressed as a SCORE constructor with scoring function $f : \mathrm{dom}(A) \to \mathbb{R}_0^+$. Given such a function, $x <_P y$ holds for $f(x) > f(y)$.

There are several sub-constructors of SCORE: In the *numerical* AROUND(A, z) the desired value should be z. If this is infeasible, values with minimal distance to z are preferred. The scoring function equals $f(v) = |z - v|$.

The LOWEST$(A, \inf_A)$ and the HIGHEST $(A, \sup_A)$ constructor prefer the minimum and maximum of the domain of A, where $\inf_A$ and $\sup_A$ are the infimum and supremum of $\mathrm{dom}(A)$. The scoring function equals $f(v) = v - \inf_A$ and $f(v) = \sup_A - v$.

The *categorical base preference* POS(A, M) expresses that a user has a set of preferred values, the set M. The scoring function equals $f(v) = 0$ for $v \in M$ and $f(v) = 1$ for $v \notin M$.

Example 2. *The wish for a tour with category id's 1 or 2 is expressed by a* POS *preference:* $P :=$ POS(cat_id, $\{1, 2\}$). *The preference selection* $\sigma[P](\text{tour})$ *on the tour dataset in Table 1 returns tours with id's* $\{1,2\}$.

2.2 Complex Preference Constructors

Complex preferences determine the relative importance of preferences and combine base or again complex preference constructors. Intuitively, people speak of "this preference is more important to me than that one" or "these preferences are all equally important".

Hence we need a notion of equality w.r.t. a preference. For a base preference P with scoring function $f(v)$ we define $\sim_P$ as the equivalence relation induced by $f(v)$. This is called *regular SV-semantics* in [Kießling, 2005].

Definition 1 (Pareto)**.** *In a* Pareto preference $P := P_1 \otimes P_2 = (A_1 \times A_2, <_P)$ *all preferences* $P_i = (A_i, <_{P_i})$ *are of equal importance, i.e., for two tuples* $x = (x_1, x_2)$, $y = (y_1, y_2) \in \mathrm{dom}(A_1) \times \mathrm{dom}(A_2)$ *we define:*

$$
\begin{aligned}
&(x_1, x_2) <_P (y_1, y_2) \iff \\
&\quad (x_1 <_{P_1} y_1 \wedge (x_2 <_{P_2} y_2 \vee x_2 \sim_{P_2} y_2)) \vee \\
&\quad (x_2 <_{P_2} y_2 \wedge (x_1 <_{P_1} y_1 \vee x_1 \sim_{P_1} y_1)) \\
&(x_1, x_2) \sim_P (y_1, y_2) \iff x_1 \sim_{P_1} y_1 \ \wedge \ x_2 \sim_{P_2} y_2
\end{aligned}
$$

If a Pareto preference P only consists of LOWEST/HIGHEST constructors, then P coincides with the traditional *Skyline query* (cf. [Börzsönyi *et al.*, 2001]), which are based on MIN/MAX. The BMO-set (Equation 1) of a Pareto preference query P is generally referred to as *the Skyline of* P.

Definition 2 (Prioritization)**.** *In a* Prioritization preference $P := P_1 \,\&\, P_2$ *the preference* $P_1 = (A_1, <_{P_1})$ *is more important than* $P_2 = (A_2, <_{P_2})$*, i.e.*

$$
\begin{aligned}
&(x_1, x_2) <_P (y_1, y_2) \iff \\
&\quad x_1 <_{P_1} y_1 \vee (x_1 \sim_{P_1} y_1 \wedge x_2 <_{P_2} y_2) \\
&(x_1, x_2) \sim_P (y_1, y_2) \iff x_1 \sim_{P_1} y_1 \ \wedge \ x_2 \sim_{P_2} y_2
\end{aligned}
$$

This is also known as lexicographical order.

3 Preferences on Hierarchies

In this section we define our new constructors for preferences on hierarchies. In the following we differ between two types of tables: The *data table* containing the tuples, ordered by some hierarchy and the *hierarchy table* containing the hierarchy itself. Table 1 exemplifies the first one whereas Table 2 is an example for the latter one.

3.1 Transitive Closure

At first we will give some preliminary definitions for dealing with hierarchies.

Let $R(x, y)$ a hierarchy table. We assume that $x \in \mathbb{N}$ is a primary key of every node (e.g., every category) where the root of the hierarchy has key 0. The attribute y points to a parent node $(y, z) \in R$ for some $z \in \mathbb{N}$. We further assume that all root nodes fulfill $y = 0$.

We interpret R as a binary relation and define powers of R:

$$
\begin{aligned}
R^0 &:= \text{identity} = \{(x, x) \mid x \in \mathbb{N}\} \\
R^{i+1} &:= \{(x, z) \mid \exists y : (x, y) \in R^i \ \wedge \ (y, z) \in R\}
\end{aligned}
$$

The transitive closure of an element x is given by

$$
\mathcal{T}(y) = \bigcup_{i \geq 0} \{x \in \mathbb{N} \mid (x, y) \in R^i\}
$$

This returns the node y (because $(y, y) \in R^0$) and all of its children in the hierarchy, e.g., elements x with $(x, y) \in R^1$ are the direct successors of y.

To let the Preference SQL interpreter know how to calculate the transitive closure we define the following directive:

```
DEFINE CLOSURE [attribute] IN [table] AS
  [hierarchy selection]
```

There

- `[attribute]` refers to the attribute for which a hierarchy exists, i.e. the foreign key of the data table referring to the hierarchy table,
- `[table]` is the data table containing this attribute,
- `[hierarchy selection]` is the SQL result specifying the hierarchy. The first attribute of the result is the primary key of the node in the hierarchy. The second attribute points to the parent key.

We give an example for our use case.

Example 3. *Let* category(id, parent_id) *be the hierarchical table and* tour(id, cat_id) *the data table (according to Tables 1 and 2; some attributes are dropped). Then the corresponding directive is*

```
DEFINE CLOSURE cat_id IN tour AS
 (SELECT id, parent_id FROM category);
```

The calculation of the transitive closure is the task of the preference language interpreter. In this sense we follow the declarative approach of SQL. That means that we only describe "what to compute" and not "how to compute" the transitive closure.

A middleware like Preference SQL [Kießling *et al.*, 2011] can do this with the help of built-in functionality of the underlying database. For example, with Oracle and some other popular databases, having a similar syntax, the transitive closure for a key with id `x` can be calculated by

```
SELECT id
FROM cat
START WITH id = x
CONNECT BY parent_id = PRIOR id;
```

3.2 New Preference Constructors

To handle hierarchies in preference algebra [Kießling, 2002], we define two preference constructors. First, we define the SUCCEEDING(A, x) constructor by

$$\text{SUCCEEDING}(A, x) = \text{POS}(A, \mathcal{T}(x))$$

where $\mathcal{T}$ is defined according to the hierarchy table given in the **`DEFINE CLOSURE`** directive for the attribute A and the table of this attribute. Note that it is the task of SQL interpreter to infer the correct table for the attribute A (and to throw an error, if this is not possible).

With the SUCCEEDING preference we formulate the wish that the attribute value of A should be x or a child of x w.r.t. the hierarchy table.

To define a preference for a certain category or, less important, for one of its parents, we declare:

$$\text{PRECEDE\_NEAR}(A, x) = \bigmathbin{\&}_{i=0}^{n} \text{POS}(A, \{\mathcal{T}(y) \mid (y, x) \in R^i\})$$

where R is the hierarchy table associated with attribute x.

The best tuples w.r.t. this preference are those tuples where A is contained in the successors of x, i.e., the set $\mathcal{T}(x)$ which is also optimal w.r.t. the SUCCEEDING(A, x) preference. Next, we go one step upwards in the hierarchy, i.e., consider y with $(x, y) \in R$, and calculate the closure $\mathcal{T}(y)$. Then the second best tuples are retrieved when A is within this set. For the third best tuples we go two steps upward, and so on.

The PRECEDE_NEAR(A, x) preference is a refinement of SUCCEEDING(A, x). If there exist tuples with $A \in \mathcal{T}(x)$ then they are returned. If not, the SUCCEEDING preference considers all tuples to be equally good. In the contrary, PRECEDE_NEAR selects those tuples which are more similar to x w.r.t. going up the hierarchy and calculating the transitive closure in every step.

Example 4. *Assume we are interested in long cycle tours, i.e., activities with category id 3. If not feasible, we search for related categories. For the preference term we obtain:*

$$\text{PRECEDE\_NEAR}(\textit{cat\_id}, 3) = \text{POS}(\textit{cat\_id}, 3) \mathbin{\&} \text{POS}(\textit{cat\_id}, \{1, 2, 3\}) \mathbin{\&} \text{POS}(\textit{cat\_id}, \{0, ..., 7\})$$

Note that the prioritization chain in the definition of SUCCEEDING is equivalent to a LAYERED preference in [Kießling, 2002]. Each preference corresponds to one layer.

3.3 A Naive Approach

A naive "straight forward" implementation of the SUCCEEDING preference simply maps $\mathcal{T}(x)$ to the explicitly calculated transitive closure. A query like

```
SELECT *
FROM tour
PREFERRING cat_id SUCCEEDING 1;
```

could be rewritten (using Oracle specific functionality) as

```
SELECT *
FROM tour
PREFERRING cat_id IN (
  SELECT id FROM cat
  START WITH id = 1
  CONNECT BY parent_id = PRIOR id );
```

Such an approach is rather inefficient. In any query occurring `attr SUCCEEDING x` and `attr SUCCEEDING y` where x is a child of y w.r.t. the hierarchy (or vice versa) results will be recalculated. If there are many of these preferences this lead to massive recalculations.

A database optimizer could exploit its cache mechanism to avoid recomputation of the transitive closure on hierarchies. However, when the cache will be cleared, the transitive closure must be computed again. Our method allows to compute the transitive closure once, in a temporary or persistent manner, and can be used when necessary.

4 Use Case

In this section we describe some real use cases in the field of tourism industry.

In the field of tourism research questionnaires are always looming at the horizon to identify stereotypes and their features. Gibson and Yiannakis [Gibson and Yiannakis, 2002] have used this technique to classify clusters defined on psychological needs as well as sociological data and to denote the clusters according to a typology of tourist roles as e.g., "Active Sport Tourist". The questionnaire originally was developed by [Yiannakis, 1986] and is called "Tourist Role Preference Scale".

To bridge the gap between stereotypes and preferences, this questionnaire may be augmented by two further statements as

a) Which activities are your favorites?

b) Which activities are the most boring for you?

The answers are not restricted by any controlled vocabulary and are mapped to a *hierarchy of concepts about activities* as shown in Figure 1 which is used by a German tourism service provider. Hierarchies represent taxonomies which classify concepts according to some features. Generalization and specialization are defined by the direction of the traversal.

Let's assume that the evaluation of the questionnaire has produced great significance that the "Active Sport Tourist"

- *likes* "CycleTour", "Dancing", "ViaFerrata", and
- *dislikes* "eBikeTour".

Not having a controlled vocabulary, fans of dancing have to be disappointed in our hierarchy. On the other hand, the touristic infrastructure is evolving over time resulting in upgrades of the hierarchy (e.g., "WingsuiteFlying"). Having no complete knowledge, the system should be based on an open-world assumption with which preferences can tackle as shown by [Endres *et al.*, 2012].

Based on the previous mentioned data table (Table 1) and hierarchy table (Table 2) as well as the above explanation, we now present some concrete examples for our new preference constructors on hierarchies. The use cases can be reproduced with a prototype implementation in [Roocks, 2015].

Example 5. *We start with modeling the preference for the "Active Sport Tourist" as given above. As we have no information about the relative importance of his wishes, the Pareto composition is the canonical choice.*

```
SELECT id, name
FROM tour
PREFERRING          cat_id SUCCEEDING 1
           AND      cat_id SUCCEEDING 5
           AND NOT (cat_id SUCCEEDING 2);
```

With that query we search for Via ferrata (id 5) all kinds of cycle tours (id 1 and its children) but we avoid eBike tours. The result will be the id's 1 (By bike around the lake) and 4 (Via ferrata – not for beginners).

Example 6. *We are now interested in a HikingTour. Less important to this we want to avoid a PilgrimPath. Anyway, we prefer a LongPilgrimPath, even though it is a child of the category PilgrimPath.*

```
SELECT id
FROM tour
PREFERRING cat_id SUCCEEDING 4
  PRIOR TO (     NOT(cat_id SUCCEEDING 6)
             AND      cat_id SUCCEEDING 7  );
```

In this example we use prioritization (`PRIOR TO`*) to express that we prefer any HikingTour over a PilgrimPath. Since we like a LongPilgrimPath we construct a Pareto preference with* `AND`*. The result is given by the id's 3, 4, and 6.*

In the following we will consider the PRECEDE_NEAR preference to search for categories and, if not feasible, their nearest predecessors.

Example 7. *Assume that we want to find a LongCycleTour (category id 3) or something similar, if this is not feasible. In the tour table there is a cycle tour (id 1, "By bike around the lake") but no long cycle tour. We state the following query:*

```
SELECT id
FROM tour
PREFERRING cat_id PRECEDE_NEAR 3;
```

The most related categories in the hierarchy are 1 (cycle tours, which is the parent category) and 2 (eBike tours, which is a child of cycle tours). Hence the query returns the tours with id's 1 and 2, as these tours reside in those categories.

Now we study the interplay with other user wishes, e.g., preferences on the difficulty.

Example 8. *We search for an activity related to a Via Ferrata. Equally important, the tour should be the difficulty "easy". These wishes tend to be conflicting, as the only Via Ferrata in the tour table is difficult. We formulate the following query:*

```
SELECT id
FROM tour
PREFERRING cat_id PRECEDE_NEAR 5
           AND difficulty = 'easy';
```

This query returns optimal compromises. The tour 4 ("Via ferrata – not for beginners") is returned, as it is optimal w.r.t. to the PRECEDE_NEAR *preference. The tours 5 and 6 are also returned. These are (long) pilgrim paths, hence they are in the second level w.r.t. to the* PRECEDE_NEAR *preference. But the difficulty is "easy", hence they are Pareto-optimal for this query. The result is* $\{4, 5, 6\}$.

Tour 2 (eBike test track), having difficulty "easy", is not in the result set, because it is Pareto-dominated by the pilgrim paths w.r.t. the given PRECEDE_NEAR *preference and hierarchy.*

5 Related Work

Hierarchical attributes are a common phenomenon in search applications providing full-text search either as middleware on top of a DBMS (see Apache Lucene[3]) or as DBMS extension (see Oracle Text[4]). Here, querying and indexing processes are enhanced via defined thesauri or lexical databases such as WordNet (see [Miller, 1995]). However, the addition of broader, narrower, or similar terms to a search query is implemented as strict query extension, meaning that scoring algorithms still follow a Boolean model and do not distinguish whether an exact concept or related terms have been found. To overcome these limitations, a similarity measure between concepts of an hierarchy has to be defined. The authors of

[3] https://lucene.apache.org

[4] https://www.oracle.com/database

[Rodríguez and Egenhofer, 2003] propose a semantic similarity measure between ontology entities. It is based on common as well as distinguishing features and a semantic neighborhood defined by distance in the relationship graph. This similarity measure allows the expression of asymmetric similarity so that a class is more similar to its superclass than vice versa. The basic elements of similarity measures for ontologies are identified in [Harispe *et al.*, 2014] to serve as base for a unifying similarity framework. Core elements include the mapping of a concept to its semantic representation in the ontology, the specificity of concepts, the commonalities of concepts, and the differences of concepts. The first two mentioned elements incorporate the hierarchy of relationships between concepts while the latter are defined on set-based operators. These elements serve as input to compute an overall similarity with the help of established measures such as the ratio model of [Tversky, 1977]. These findings are the base for numerical approaches in the field of decision support. In order to define similarity within a taxonomy of different cuisines of restaurants, the authors of [Espeter and Raubal, 2009] propose a similarity measure based on distance within the hierarchy. Similarity is at its maximum if a desired concept subsumes an investigated concept and at its minimum if two concepts only share the root concept. Here, asymmetry is defined in such a way that a super-concept is considered equal to its sub-concepts but not vice versa.

In the domain of databases, preferences on hierarchical attributes are proposed in [Lukasiewicz *et al.*, 2013] following a Datalog+/- approach. Given an ontology together with a specific database instance, both unconditional and conditional queries can be expressed. Besides queries on single attributes, Skyline and generalized Top-k queries are admissible. In [Golfarelli and Rizzi, 2009b], OLAP preferences are defined. A hierarchy is induced by different levels of aggregation via Group-by sets as part of the OLAP process. Base preferences such as COARSEST, FINEST, CONTAIN, and NEAR are specified to express a preference for a specific level of aggregation within that hierarchy. Besides these specific approaches, general preference models for categorical attributes known from research in decision support, databases, or AI are applicable as long as a hierarchy is small enough to allow an explicit expression of the preference order. By this means, any method for preference expression, from database queries over CP-nets to logical formulas, is applicable.

6 Summary and Outlook

In this workshop paper we have addressed the problem of preference database queries over hierarchies. We introduced a technique to handle hierarchies which extends preference algebra and can easily be integrated in preference query languages. For this we defined the SUCCEEDING and PRECEDE_NEAR constructors and extended the syntax of Preference SQL. Our approach allows the simple definition of a transitive closure in database systems and overcomes the massive recomputation when using standard methods.

A database optimizer can previously check which category id's occur in the SUCCEEDING and PRECEDE_NEAR preferences. Based on this the optimizer can find a minimal set of nodes for which the transitive closure has to be computed to answer the query. Any recomputation of subsets of the hierarchy graph can be avoided. Another optimization strategy could be to calculate the transitive closure of all elements according to the **`DEFINE CLOSURE`** directive which is valid for the entire session. This implies that evaluating the SUCCEEDING and PRECEDE_NEAR preferences in the following queries can be done very efficiently.

With our approach, wishes on categories can be seamlessly connected with all other kinds of preferences. As we illustrated in the use cases, user preference for a certain kind of outdoor activity, the difficulty or other attributes like the geographical region can be handled in the same systematical manner. Compared to more sophisticated methods of similarity and distance measures on hierarchies, the results of a query can be easily comprehended by the user without the existence of "hidden factors".

One could use the preferences introduced in this paper also to combine user wishes in different kinds of hierarchies. Assume a data set of outdoor activities with an additional attribute for the location of the activity within a geographical hierarchy, e.g., country, region, district, town, etc. In a straight forward way we could express a search for "mountainbike tours within the Tannheim valley" by a Pareto combination of two SUCCEEDING preferences for these preferred categories. A PRECEDE_NEAR preference on a small village would be similar to a *NEARBY* preference (cf. [Wenzel *et al.*, 2012]) but respects the borders of regions and districts. This might by helpful for tourists using ropeways or public transport, because the validity of week/saison passes for tourist transportation is usually subject to certain regions.

For the future, further simplifications on the language level are possible, e.g., allowing the category names instead of the id's in the constructor of the hierarchical preferences. In the present work we kept the approach clear and simple to obtain a readable formalism and a simple prototype implementation based on the rPref package. For the future we plan to implement this kind of preference handling on hierarchies in the Preference SQL system.

Acknowledgments

This work has been partially funded by the Bavarian Ministry of Economic Affairs, Infrastructure, Transport and Technology, grant number IUK-1307-0004//IUK434/003.

References

[Börzsönyi *et al.*, 2001] S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *ICDE '01: Proceedings of the 17th International Conference on Data Engineering*, pages 421–430, 2001.

[Chomicki, 2003] J. Chomicki. Preference Formulas in Relational Queries. In *TODS '03: ACM Transactions on Database Systems*, volume 28, pages 427–466, 2003.

[Endres *et al.*, 2012] M. Endres, P. Roocks, F. Wenzel, A. Huhn, and W. Kießling. Handling of NULL Values in Preference Database Queries. In *MPref '12: Proceedings of the 6th Multidisciplinary Workshop on Advances in Preference Handling*, August 2012.

[Espeter and Raubal, 2009] Martin Espeter and Martin Raubal. Location-based decision support for user groups. *Journal of Location Based Services*, 3(3):165–187, 2009.

[Gibson and Yiannakis, 2002] H. Gibson and A. Yiannakis. Tourist roles: Needs and the Lifecourse. *Annals of Tourism Research*, 29(2):358 – 383, 2002.

[Golfarelli and Rizzi, 2009a] M. Golfarelli and S. Rizzi. Expressing OLAP Preferences. In *Proceedings of SSDBM '09*, pages 83–91, 2009.

[Golfarelli and Rizzi, 2009b] Matteo Golfarelli and Stefano Rizzi. Expressing OLAP Preferences. In *21st International Conference on Scientific and Statistical Database Management SSDBM*, pages 83–91, New Orleans, LA, USA, 2009.

[Harispe *et al.*, 2014] Sébastien Harispe, David Sánchez, Sylvie Ranwez, Stefan Janaqi, and Jacky Montmain. A framework for unifying ontology-based semantic similarity measures: A study in the biomedical domain. *Journal of Biomedical Informatics*, 48:38–53, 2014.

[Kaci, 2011] Souhila Kaci. *Working with Preferences: Less Is More*. Springer, 2011.

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

[Kießling, 2002] W. Kießling. Foundations of Preferences in Database Systems. In *VLDB '02: Proceedings of the 28th International Conference on Very Large Data Bases*, pages 311–322. VLDB Endowment, 2002.

[Kießling, 2005] W. Kießling. Preference Queries with SV-Semantics. In *In Proceedings of COMAD '05*, pages 15–26, 2005.

[Lukasiewicz *et al.*, 2013] Thomas Lukasiewicz, Maria Vanina Martinez, and Gerardo I. Simari. Preference-Based Query Answering in Datalog+/- Ontologies. In *23rd International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1017–1023, Beijing, China, 2013.

[Mandl *et al.*, 2015] S. Mandl, O. Kozachuk, M. Endres, and W. Kießling. Preference Analytics in EXASolution. 16th Conference on Database Systems for Business, Technology, and Web. 2015.

[Miller, 1995] George A. Miller. WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[Rodríguez and Egenhofer, 2003] M. Andrea Rodríguez and Max J. Egenhofer. Determining Semantic Similarity Among Entity Classes from Different Ontologies. *IEEE Transactions on Knowledge and Data Engineering*, 15(2):442–456, 2003.

[Roocks and Kießling, 2013] P. Roocks and W. Kießling. R-Pref: Rapid Prototyping of Database Preference Queries in R. In *DATA 2013*, pages 104–111, 2013.

[Roocks, 2015] P. Roocks. *R Script containing examples from the paper*, 2015. `http://www.p-roocks.de/mpref-hierarchies.r`.

[Rossi *et al.*, 2011] Francesca Rossi, Brent Venable, and Toby Walsh. *A Short Introduction to Preferences: Between Artificial Intelligence and Social Choice*. Morgan & Claypool, July 2011.

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

[Tversky, 1977] Amos Tversky. Features of Similarity. *Psychological Review*, 84(4):327–352, 1977.

[Wenzel *et al.*, 2012] F. Wenzel, M. Endres, S. Mandl, and W. Kießling. Complex Preference Queries Supporting Spatial Applications for User Groups. *PVLDB*, 5(12):1946–1949, 2012.

[Yiannakis, 1986] A. Yiannakis. The Ephemeral Role of the Touris: Some Correlates of Tourist Role Preference. In *Paper presented at the NASSS Confrence, Las Vegas, Nevada*, 1986.