Complex Preference Queries Supporting Spatial Applications for User Groups

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

ABSTRACT

Our demo application demonstrates a personalized location-based web application using *Preference SQL* that allows single users as well as groups of users to find accommodations in Istanbul that satisfy both hard constraints and user preferences. The application assists in defining spatial, numerical, and categorical base preferences and composes complex preference statements in an intuitive fashion. Unlike existing location-based services, the application considers spatial queries as soft instead of hard constraints to determine the best matches which are finally presented on a map. The underlying Preference SQL framework is implemented on top of a database, therefore enabling a seamless application integration with standard SQL back-end systems as well as efficient and extensible preference query processing.

1. INTRODUCTION

Spatial applications such as location-based services aim to provide new services to users based on the knowledge of their current or future locations. The search for a suitable accommodation in Istanbul during the VLDB conference is a typical scenario. In this case, users want to find the "best" accommodation according to their current preferences and projected location. However, existing location-based query applications are reducing the meaning of "best" to "closest" in terms of distances. If desired, preferences are only applied as postfilter to the returned location-based result. Additionally, location-based applications often allow a spatial window search, to avoid the "shortest distance" problem. However, existing applications treat this restriction as a hard constraint. Thus, if no accommodation fulfills the users preferences the empty result effect occurs.

Preference query processing mitigates the described drawbacks and has become an important concept in areas such as multi-criteria decision-making, computational social choice theory, operations research, artificial intelligence, and personalized databases (see [6] for an overview). Focusing on databases, conventional search queries via SQL only support hard constraints with an exact-match semantics. However, people also include preferences in their decisions. These preferences are like soft constraints, requiring a match-making process instead: *If my favorite choice is available in the database, I will take it. Otherwise, instead of getting nothing, I am open to alternatives, but show me only the best ones available in the database.* Therefore, improving SQL-based search capabilities asks for extending SQL query technology towards a preference model, which of course should be powerful, flexible and intuitive for the user, but simultaneously performance restrictions must be met.

In recent years, many preference handling approaches have been proposed and implemented in a different manner, e.g. the recently published FlexPref/CareDB framework [3]. It enables extensible preference query processing based on preferences which must be integrated into the PostgreSQL database system using a kind of plugin function. CareDB provides a personalized location-based service to users based on their preferences and current surrounding context. However, CareDB lacks the means for specifying distinguished spatial preferences, i.e. "best" still means "closest" regarding location. Even with the possibility to define user preferences inside of FlexPref/CareDB and the support of a large variety of preference queries, FlexPref doesn't implement the full SQL standard which is necessary for convenient query specification. Furthermore, FlexPref does not provide some of the most intuitive preferences, e.g., it is not possible to say that one preference is more important than another.

In this demo paper we demonstrate how to use Preference SQL for a location-based search of accommodations in the city of Istanbul. Our demo application allows single users as well as groups of users to specify complex spatial preference queries. We support the full SQL 92 standard, e.g., joins, group by, subqueries, etc. Preference SQL provides many intuitive and inductive constructors to specify numerical, categorical, and spatial preferences which can be combined with *equal importantance* (Pareto) or *ordered importance* (Prioritization). Furthermore, preferences can be *ranked* using pre-defined and self-defined utility functions, or reversed, cp. [2]. Moreover, Preference SQL supports context-aware preference queries, too [4].

The rest of the paper is organized as follows: Section 2 provides an overview of the Preference SQL framework, the query language and the demo architecture. Section 3 covers our demonstration and provides some uses cases which demonstrate how to use Preference SQL for a preference location-based real-life application scenario.

Proceedings of the VLDB Endowment, Vol. 5, No. 12

2. PREFERENCE SQL OVERVIEW

Preference SQL is a declarative extension of standard SQL by strict partial order preferences, behaving like soft constraints under the *Best-Matches-Only* (BMO) query model. The result of a preference query consequently contains all tuples of the database which are not dominated by any other tuple concerning the users preferences, cp. [1]. Preference queries can be formulated intuitively following an inductive constructor-based approach using an extension of SQL, cp. [2] and ***http://www.trial.PreferenceSQL.com***.

2.1 The Preference SQL Language

Syntactically, Preference SQL [2] extends the SELECT statement of SQL by an optional *PREFERRING* clause leading to the following schematic design:

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

Figure 1: Preference SQL query block

The keywords SELECT, FROM, WHERE, GROUP BY, HAVING, and ORDER BY are treated as standard SQL keywords. The *PREFERRING* clause specifies a preference by means of the preference constructors given in [2]. For example, the wish for an accommodation which has between 50 and 100 bedrooms can be specified by *'number_rooms BETWEEN 50, 100'*, if *number_rooms* is the attribute of a database relation. The *'room_rate LESS THAN 150'* preference expresses the wish for a rate less than 150 Euros (analogously *MORE THAN*). *CONTAINS* is useful for a preference based full text search, e.g., *amenities CONTAINS 'Swimming Pool AND Internet'*. The *DUAL* preference constructor reverses the order of the preference. Furthermore, the Pareto preference (combined by the *AND* keyword in the *PREFERRING* clause) treats two or more preferences as equally important, whereas for Prioritization (*PRIOR TO*) one preference is more important than another. Further keywords such as *GROUPING* are provided to modify preference evaluation, *BUT ONLY* for the definition of postfilters, and *TOP* and *LIMIT* to regulate the number of results.

After specifying a preference it is evaluated on the result of the hard conditions stated in the WHERE clause, returning the BMO-set. Empty results can only occur if the WHERE clause returns an empty result.

2.2 The Preference SQL Demo Architecture

Preference SQL is a Java 1.6-based on top of the database approach, cp. Figure 2. This approach is convenient to implement, as it exists in a separate code based outside the database. For a seamless application integration we have extended standard JDBC technology towards Preference SQL JDBC. This enables the client application to submit Preference queries through familiar SQL clients. This approach has proven its flexibility and efficiency in various prototype applications, e.g. [4].

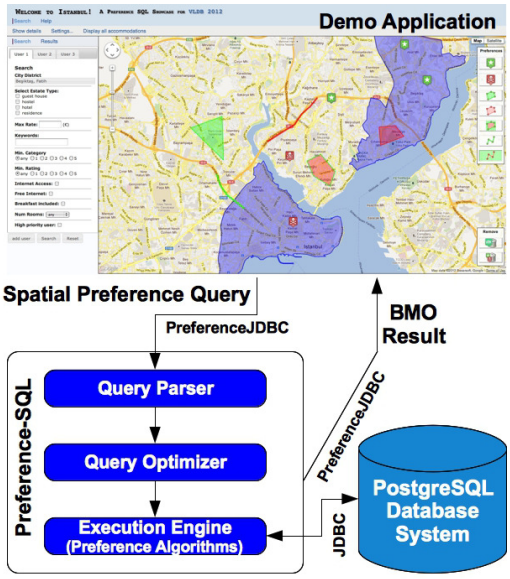

Figure 2: Architecture of our demo application

Our demo application works as follows: The user specifies preferences and hard constraints in a web-based application to find the best accommodations in Istanbul. In a next step, the demo application generates a Preference SQL query which is sent to the Preference SQL system. The query gets parsed and transformed into an initial operator tree. Afterwards, we apply heuristics for preference algebraic operator tree optimization (see [2] for details). Likewise guided by a cost-based model, the Preference query optimizer determines suitable sub-trees of the final optimized operator tree for offloading them to the back-end SQL system. Those sub-trees are retranslated into SQL and sent via JDBC to the attached database. For evaluation of the preferences, several efficient algorithms have been implemented in our Preference Execution Engine, cp. [2].

2.3 Spatial Preferences in Detail

To demonstrate the extensibility of Preference SQL towards new application domains, the showcase application highlights a new category of spatial preference constructors as *novel* research contribution of this paper. Spatial constructors let users define geometries towards which they have some kind of preference. For preferences concerning geometries of type point, the longitude and latitude are provided to the constructor. For all other types of geometries, spatial constructors are accepting a KML string as argument defining the geometry in question. For geometries of type polygon, KML provides the notation

```
'<Polygon><outerBoundaryIs><LinearRing>
<coordinates>...</coordinates>
</LinearRing></outerBoundaryIs></Polygon>'
```

and for linestrings accordingly. The use of KML as a widely accepted standard for the definition and presentation of spatial data enhances the interoperability with leading spatial web applications, APIs and virtual globes.

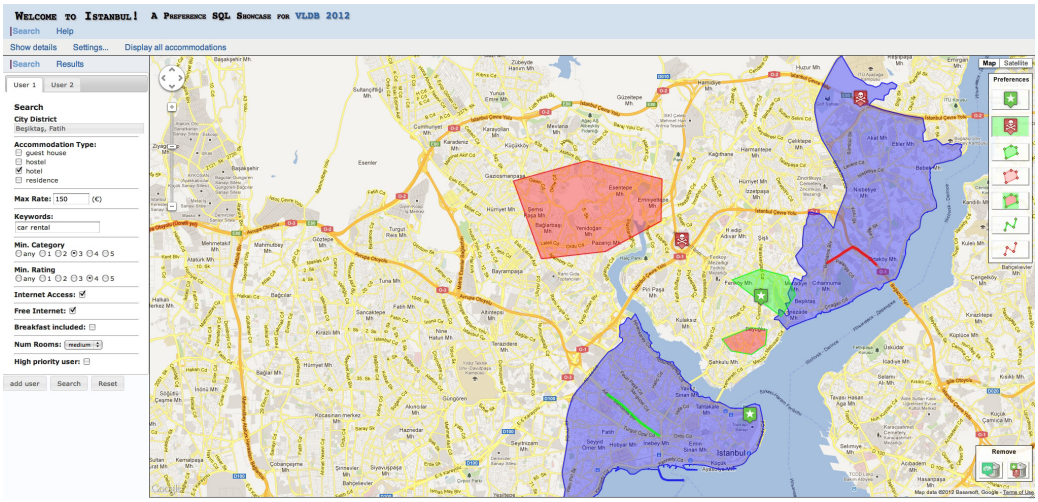

Figure 3: Application screenshot

We provide the following spatial preference constructors:

- ▷ **NEARBY** defines a preference for geometries that are close to a user specified point. In combination with the DUAL constructor, a preference for geometries that are farthest away from a specified point can be expressed.
- ▷ **WITHIN** defines a preference for geometries that are within or close to a user specified polygon. In combination with DUAL, geometries farthest away from a specified polygon can be expressed.
- ▷ **ONROUTE** defines a preference for geometries that are on or close to a defined linestring. In combination with DUAL, geometries that are farthest away from a specified linestring can be expressed.
- ▷ **BUFFER** defines a preference for geometries that are not within but close to a defined polygon.

Spatial preferences are evaluated on top of a spatially extended database, e.g. PostgreSQL with PostGIS, using implemented distance functions as defined by the SQL/MM Spatial Standard such as ST_Distance to determine the preference order.

3. SHOWCASE APPLICATION

Our demonstration scenario showcases a web application based on Preference SQL that allows single users as well as groups of users to find accommodations in Istanbul that satisfy user preferences in the best way possible. The application assists in defining spatial, numerical, and categorical base preferences in an intuitive fashion and composes complex preference statements according to a predefined query composition. The generated Preference SQL statement is evaluated using a database of accommodations in Istanbul to determine best matches which are finally presented on a map. Data for the application scenario has been generated from an OpenStreetMap dump of the Istanbul area including administrative boundaries augmented by information from tourist websites such as *tripadvisor.com* and *booking.com*.

3.1 Application Details

The user interface (cp. Figure 3) consists of two main areas: The left-hand side serves as input for numerical and categorical preferences as well as hard constraints concerning the type of accommodation. Furthermore, a drop-down menu allows the selection of preferred districts of the Istanbul region. The search process can be augmented from a single-user to a multi-user scenario by adding search tabs via the "add user" button. The right-hand side of the screen holds a map of Istanbul per user on which spatial preferences mentioned in Subsection 2.3 can be individually defined by clicking the corresponding preference symbol and drawing the preferred geometry onto the map (cp. Figure 4). By switching search tabs on the left-hand side, the map adapts accordingly and only displays the personal spatial preferences of the selected user. Preferred districts are also visualized in these maps. The generated Preference SQL statement can be investigated by using the detailed view.

3.2 Use Case Scenarios

To highlight the functionality of Preference SQL in the spatial domain we are introducing two use cases. The first one illustrates the process of retrieving suitable accommodations for a single user while the second one describes a search scenario for a group of users. Explicit KML is stripped from Preference SQL queries to increase readability.

use case 1: single user scenario
Bob is looking for a small accommodation in Istanbul during his attendence of the VLDB conference. He prefers a customer rating of at least three stars and a location close to the Archeological Museum.

Bob's base preferences are collected by the application and combined via complex preference constructors. In case all preferences are of equal importance, they are translated into the following Preference SQL statement:

```
SELECT * FROM accommodation
PREFERRING number_rooms BETWEEN 1,50 AND
 customer_rating MORE THAN 3 AND
 location NEARBY 28.98162, 41.01102
```

Figure 4: Input of NEARBY, WITHIN, ONROUTE with DUAL preferences, and BUFFER

In this case only Pareto is used as complex constructor and NEARBY is the only spatial preference. The query thus defines a modified Location-Dependent Skyline Query as formulated by [7] with a runtime of *50 ms* in our application. However, other complex constructors such as Prioritization are available for query composition. Additionally, Bob might prefer to stay in the 'Fatih' district of Istanbul leading to preferences on complex spatial objects. If he is also interested in 'bike rentals' during his stay then textual preferences are added to the search. These changes to the first query are included into the following statement:

```
SELECT * FROM accommodation
PREFERRING
 (location WITHIN 'KML' AND
  location NEARBY 28.98167, 41.01111)
 PRIOR TO
 (number_rooms BETWEEN 1, 50 AND
  customer_rating MORE THAN 3 AND
  amenities CONTAINS 'bike rentals')
```

The generated query with a runtime of *42 ms* now contains a WITHIN preference for a polygon defining the administrative boundary of the 'Fatih' district as retrieved from the integrated OpenStreetMap data. The textual preference constructor CONTAINS is applied to a string attribute.

In a second use case we describe what happens if Bob's boss Paul decides to join him on the journey to Istanbul. Now both are looking for a common accommodation that satisfies each user's preferences in the best way possible.

use case 2: group scenario
Paul only accepts hotels as suitable form of accommodation. He likes to stay nearby the Bosphorus Bridge but also close to the conference venue. The hotel should not be directly located in the Taksim Square area but close to it. The room rate should be less than 95 Euros including breakfast.

Before constructing the group search query, Paul's individual preferences are investigated, leading to the following Preference SQL query with a runtime of *31 ms*:

```
SELECT * FROM accommodation
WHERE type IN ('hotel')
PREFERRING
 (location ONROUTE 'KML' AND
  location NEARBY 28.98845, 41.04435 AND
  location BUFFER 'KML')
 PRIOR TO
 (room_rate LESS THAN 95.0 AND
  breakfast IN ('included'))
```

The type of accommodation is a hard constraint and thus appears in the WHERE clause. Spatial preferences are expressed by a ONROUTE preference containing the coordinates of points on the Bosphorus Bridge, a NEARBY preference with the coordinates of the conference venue and a BUFFER preference with a polygon around Taksim Square.

These preferences have to be joined with Bob's final preferences of use case 1. In order to form a common group preference term, single user preferences are combined via complex constructors. By choice of Pareto for equal importance of users or Prioritization for ordered importance, group hierarchies can be induced. Since Paul checked the "high priority user" option, his preferences are prioritized to Bob's preferences. The final Preference SQL query for Paul and Bob with a runtime of *43 ms* is defined as follows:

```
SELECT * FROM accommodation
WHERE type IN ('hotel')
PREFERRING
((location ONROUTE 'KML' AND
  location NEARBY 28.98845, 41.04435 AND
  location BUFFER 'KML')
 PRIOR TO
 (room_rate LESS THAN 95.0 AND
  breakfast IN ('included')))
PRIOR TO
((location WITHIN 'KML' AND
  location NEARBY 28.98173, 41.01112)
 PRIOR TO
 (number_rooms BETWEEN 1, 50 AND
  customer_rating MORE THAN 3 AND
  amenities CONTAINS 'bike rentals'))
```

If location-dependent skyline queries are combined via Pareto constructors, the resulting statement defines an extended Spatial Skyline Query (SSQ) as formulated by [5]. Preference SQL augments this concept by allowing all kinds of complex preference constructors as well as preferences on complex spatial objects. Therefore, Preference SQL is a comprehensive framework for preference query processing supporting numerical, categorical, and spatial preferences.

4. REFERENCES

[1] W. Kießling. Foundations of Preferences in Database Systems. In *VLDB*, pages 311–322, 2002.

[2] W. Kießling, M. Endres, and F. Wenzel. The Preference SQL System - An Overview. *DEB*, 34(2):11–18, 2011.

[3] J. J. Levandoski, M. E. Khalefa, and M. F. Mokbel. An Overview of the CareDB Context and Preference-Aware Database System. *DEB*, 34(2):41–46, 2011.

[4] P. Roocks, M. Endres, S. Mandl, and W. Kießling. Composition and Efficient Evaluation of Context-Aware Preference Queries. In *DASFAA*, pages 81–95, 2012.

[5] M. Sharifzadeh and C. Shahabi. The Spatial Skyline Queries. In *VLDB*, pages 751–762, 2006.

[6] K. Stefanidis, G. Koutrika, and E. Pitoura. A Survey on Representation, Composition and Application of Preferences in Database Systems. *ACM TODS*, 36(3): article 19, 2011.

[7] B. Zheng, K. C. K. Lee, and W.-C. Lee. Location-Dependent Skyline Query. In *MDM*, pages 148–155, 2008.