A Demonstration of FlexPref: Extensible Preference Evaluation Inside the DBMS Engine[*]

Justin J. Levandoski Mohamed F. Mokbel Mohamed E. Khalefa
Venkateshwar R. Korukanti
Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN, USA
{justin,mokbel,khalefa,venkat}@cs.umn.edu

ABSTRACT

This demonstration presents FlexPref, a framework implemented inside the DBMS query processor that enables efficient and extensible preference query processing. FlexPref provides query processing support *inside* the database engine for a wide-array of preference evaluation methods (e.g., skyline, top-k, k-dominance, k-frequency) in a *single* extensible code base. Integration with FlexPref is simple, involving the registration of only *three* functions that capture the essence of the preference method. Once integrated, the preference method "lives" at the core of the database, enabling the efficient execution of preference queries involving common database operations (e.g, selection, join). Functionality of FlexPref, implemented inside PostgreSQL, is demonstrated through the implementation and use of several state-of-the-art preference methods in a real application scenario.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems

General Terms

Algorithms, Design, Performance

1. INTRODUCTION

Preference query processing has become an important concept, helping to realize applications such as multi-criteria decision-making tools and personalized databases. In recent years, a number of preference methods have been proposed, including top-k [4], skylines [1], k-dominance [2], k-frequency [3], and top-k dominance [7]. Each of these methods have semantics that challenge the notion of "best" answers. Since "best" is subjective, we will likely see the proposal of a number of new preference methods in the future. From a systems perspective, a fundamental issue is how we can *realize* each existing and future method inside a DBMS engine in order to efficiently handle arbitrary select-project-join queries.

Currently, there exist two approaches to implementing preference methods using a DBMS. (1) The first and most common is the <u>on-top</u> approach, where the preference method is implemented outside of the DBMS. This approach treats the DBMS as a "black box", where the preference evaluation method is completely decoupled from the database, and hence not concerned with internal database operations (e.g., joins) necessary to retrieve the data. The on-top approach is convenient to implement, as it exists in a separate code-based outside the DBMS, but has limited efficiency, as it cannot interact with database internals to take advantage of query processor optimizations. (2) The second approach is the <u>*built-in*</u> approach, that tightly couples the preference method with the DBMS engine. This approach is efficient, as it pushes the preference evaluation inside the DBMS, but impractical, as it would require a custom DBMS implementation for each preference method, requiring a non-trivial amount of effort and maintenance.

This demonstration presents FlexPref [5], a centrist approach to preference implementation in a DBMS, combining the convenience of the *on-top* approach with the efficiency of the *built-in* approach. FlexPref is implemented inside the query processor of PostgreSQL [6], tightly coupled with existing DBMS operations (e.g., selection, joins), and is extensible to arbitrary preference methods. The major advantages of FlexPref are:

1. *Ease of implementation.* Placing a preference function in FlexPref (and hence within the DBMS), requires the implementation of only three functions (outside PostgreSQL) that define the semantics of the preference method, that are then registered with FlexPref.
2. *Small footprint.* FlexPref requires orders of magnitude less code to implement a preference method compared to the built-in approach. For instance, a custom skyline implementation in PostreSQL [6] requires 2,000 lines of code, while in FlexPref requires only 300 lines.
3. *Seamless DBMS integration.* Since FlexPref is extensible, a preference method, once registered, is instantly ready for use in arbitrary select-project-join queries.
4. *Efficiency.* FlexPref is tightly coupled with the DBMS query processor, meaning a registered preference method realizes efficient query processing comparable to the *built-in* implementation approach.

This demonstration showcases the registration of several state-of-the-art preference functions in FlexPref, underneath an actual location-based application built into Google Maps.

[*]This work is supported in part by the National Science Foundation under Grants IIS-0811998, IIS-0811935, CNS-0708604, and by a Microsoft Research Gift

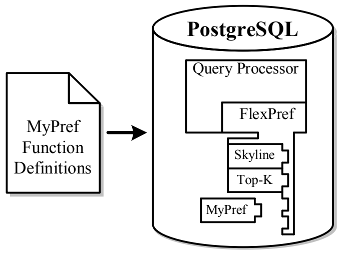


Figure 1: Flexpref

2. USING FLEXPREF

Using FlexPref is simple and straightforward. This section provides an overview of preference method registration in FlexPref, and the method's use in a DBMS.

2.1 General Functions

We require the definition of two macros and three functions in order to register a preference method with FlexPref. The two required macros are:

- `#define DefaultScore`: Each object in FlexPref is associated with a score, internal to the underlying preference method. Defining a default score ensures that each object is assigned a value.
- `#define IsTransitive`: Indicates whether the preference method exhibits transitivity or not. Knowledge of transitivity leads to better query processing efficiency.

The three general functions requiring implementation are:

- `PairwiseCompare(Object P, Object Q)`: Given two data objects P and Q, *update* the score of P and return 1 if Q can *never* be a preferred object, -1 if P can *never* be a preferred object, 0 otherwise.
- `IsPreferredObject(Object P, PreferenceSet S)`: Given a data object P and a set of preferred objects S, return *true* if P is a preferred object and can be added to S, *false* otherwise.
- `AddPreferredToSet(Object P, PreferenceSet S)`: Given a data object P and a preference set S, add P to S and remove or rearrange objects from S, if necessary.

These functions break down preference evaluation into a set of modular operations that need *not* be aware of query processor specifics.

2.2 Preference Method Registration

Adding a preference method to FlexPref requires the implementation of *three* functions outside the database engine (details in Section 2.1). Once implemented, the preference method is registered using a `DefinePreference` command:

```
DefinePreference [Name] WITH [File]
```

The *name* argument is the name of the preference method, while the *file* argument specifies the file containing the function definitions. `DefinePreference` compiles the preference code into our framework. This process is depicted in Figure 1 for a preference method "MyPref".

2.3 Queries in FlexPref

Once a preference method is registered with FlexPref, it can be used in database queries immediately. FlexPref requires the extension of the SQL syntax in order to select the appropriate preference methods and specify their objectives. FlexPref adds a `Preferring` and `Using` clause to conventional SQL in order to issue preference queries. A typical query in FlexPref is:

```
Select [Select Clause]   From [Tables]
Where [Where Clause]
Preferring [Preference Attributes]
Using [method] With [Parameter]
Objectives [Objective]
```

Here, the method (with objectives) specified in the `Using` clause is responsible for selecting the preference evaluation method to be applied over the attributes given in the `Preferring` clause. As an example, consider the following query for the well-known skyline [1] preference method implemented in FlexPref.

```
Select * From Restaurant R
Preferring R.price d1 AND R.dist d2 AND R.rating d3
Using Skyline With Objectives MIN d1, MIN d2, MAX d3;
```

This query will evaluate the skyline of restaurant data, where the preference objectives require minimizing both price and distance attributes, while maximizing rating. As another example, consider the following abbreviated query for the Top-k dominating [7] preference method.

```
Using Top-K-Domination With K=2
Objectives MIN d1, MIN d2, MAX d3;
```

Here, the `Using` clause specifies that: (1) K=2 answers are required and (2) Preference is based on minimizing both price and distance, while maximizing rating.

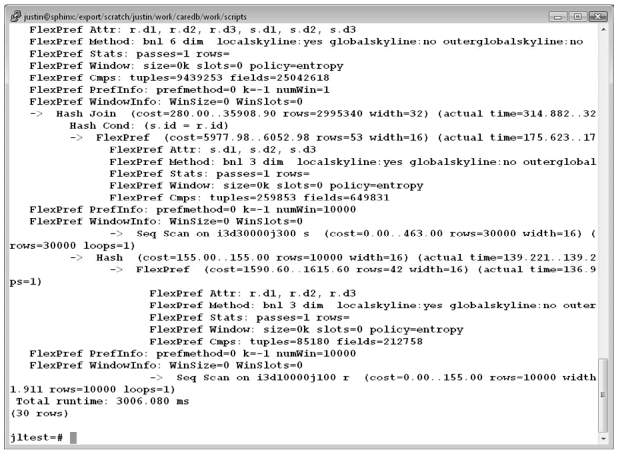
3. FLEXPREF FUNCTIONALITY

To provide efficient query processing support for the registered preference methods, FlexPref provides extensible query operators, implemented inside the engine of the DBMS. We note that these operators require implementation in the query processor *only once*. Once in place, these operators use the general macro and function definitions to evaluate the preference method from within the query processor, requiring no further changes to the DBMS engine. We now provide an overview of three core FlexPref extensible operators we have developed (details in [5]).

Selection. The FlexPref selection algorithm evaluates the set of preferred objects from a *single table*. The main idea is to compare tuples pairwise while incrementally building a preferred answer set. During execution, a data object P may be found to be dominated (i.e., guaranteed *never* to be a preferred answer). If the underlying preference method is *transitive*, P is immediately discarded and not processed further, thus leading to more efficient execution.

Join. The FlexPref join algorithm enables efficient preference evaluation for data that exists in *multiple* tables. The main idea behind this join operation involves using the general functions to *prune* tuples from the join input that are guaranteed not to be in the final answer. Pruning enhances join performance for two reasons: (a) the amount of data to be joined from input tables is greatly reduced due to pruning the input data, and (b) the amount of data processed by the final preference evaluation after the join is reduced based on the multiplying factor of the join.

Sorted list (index) access. Availability of sorted attributes (e.g., indexes such as the B+-tree), allow for efficient preference evaluation. The idea is that complete preference answer generation can be guaranteed after reading only a *portion* of the sorted data, thus reducing the I/O overhead compared to query processing over unsorted or non-indexed data. FlexPref exploits this idea by employing an algorithm capable of processing sorted attributes in round-robin fashion, and stopping I/O once a stopping condition, provided by the general functions, has been met.



```
justin@sphinx:/export/scratch/justin/work/caredb/work/scripts
   FlexPref Attr: r.d1, r.d2, r.d3, s.d1, s.d2, s.d3
   FlexPref Method: bnl 6 dim  localskyline:yes globalskyline:no outerglobalskyline:no
   FlexPref Stats: passes=1 rows=
   FlexPref Window: size=0k slots=0 policy=entropy
   FlexPref Cmps: tuples=9439253 fields=25042618
   FlexPref PrefInfo: prefmethod=0 k=-1 numWin=1
   FlexPref WindowInfo: WinSize=0 WinSlots=0
   ->  Hash Join  (cost=280.00..35908.90 rows=2995340 width=32) (actual time=314.882..32
         Hash Cond: (s.id = r.id)
         ->  FlexPref  (cost=5977.98..6052.98 rows=53 width=16) (actual time=175.623..17
               FlexPref Attr: s.d1, s.d2, s.d3
               FlexPref Method: bnl 3 dim  localskyline:yes globalskyline:no outerglobal
               FlexPref Stats: passes=1 rows=
               FlexPref Window: size=0k slots=0 policy=entropy
               FlexPref Cmps: tuples=259853 fields=649831
   FlexPref PrefInfo: prefmethod=0 k=-1 numWin=10000
   FlexPref WindowInfo: WinSize=0 WinSlots=0
               ->  Seq Scan on i3d30000j300 s  (cost=0.00..463.00 rows=30000 width=16) (
rows=30000 loops=1)
         ->  Hash  (cost=155.00..155.00 rows=10000 width=16) (actual time=139.221..139.2
               ->  FlexPref  (cost=1590.60..1615.60 rows=42 width=16) (actual time=136.9
ps=1)
                     FlexPref Attr: r.d1, r.d2, r.d3
                     FlexPref Method: bnl 3 dim  localskyline:yes globalskyline:no outer
                     FlexPref Stats: passes=1 rows=
                     FlexPref Window: size=0k slots=0 policy=entropy
                     FlexPref Cmps: tuples=85180 fields=212758
   FlexPref PrefInfo: prefmethod=0 k=-1 numWin=10000
   FlexPref WindowInfo: WinSize=0 WinSlots=0
                     ->  Seq Scan on i3d10000j100 r  (cost=0.00..155.00 rows=10000 width
1.911 rows=10000 loops=1)
 Total runtime: 3006.080 ms
(30 rows)

jltest=#
```

Figure 2: PostgreSQL backend with query plan

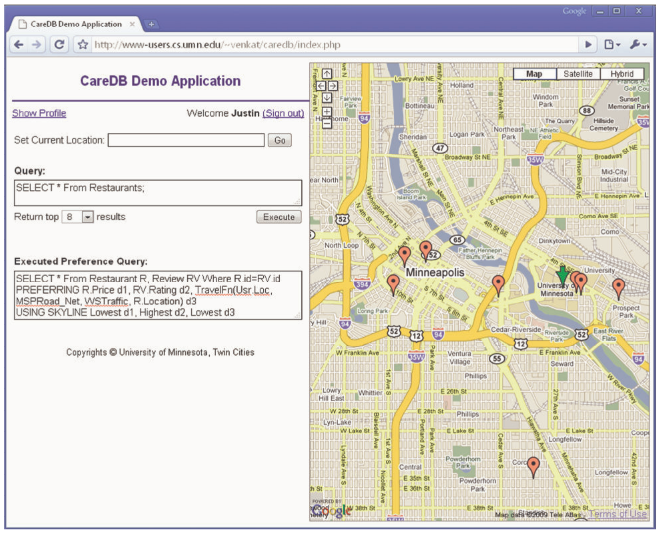
4. DEMO SCENARIO

Our demonstration scenario showcases the complete workflow for implementing and using a preference function within FlexPref. We allow the demonstration attendee to experience this workflow for any one of five popular state-of-the-art preference methods: top-k [4], skylines [1], k-dominance [2], k-frequency [3], or top-k dominance [7]. The showcase involves: (1) *Preference registration*, viewing preference method's general function implementation and registration in FlexPref, (2) *backend access*, viewing the preference method's use within our PostgreSQL prototype augmented with FlexPref, and (3) *an application scenario*, viewing FlexPref in action under an application scenario developed for this demonstration: a preference-aware location-based restaurant finder implemented with Google Maps.

Preference registration. We provide to the demo attendee the general function implementation for any one of five state-of-the-art preference functions (as outlined in Section 2.1). We then demonstrate the ease of compilation and registration of the preference function within our FlexPref prototype (as outlined in Section 2.2). The demo attendee is also welcome to create their own preference method, or alter existing methods, and (re)compile it into our FlexPref prototype.

Backend access. Once the preference method is registered with our FlexPref prototype, we then allow the demo attendee to witness the newly-registered preference method in action. We first explore the execution of FlexPref by accessing PostgreSQL directly through a backend client, as depicted in Figure 2. Attendees can: (a) Issue *ad-hoc* select-project-join preference queries that follow our new FlexPref SQL syntax outlined in Section 2.3. (b) Explore query plans used to execute preference and context-aware queries. For example, Figure 2 depicts the statistics for a query plan using the FlexPref operator for a join-query between two tables. (c) Enable or disable *FlexPref* optimization features (e.g., join optimizations) to contrast performance speedups between relatively naive query plans and more optimized plans.

Application scenario. We also showcase the use of FlexPref within a live application scenario developed for this demonstration. Our application is a location-based restaurant-finder that relies on our FlexPref prototype for its underlying DBMS preference query processing. Figure 3



CareDB Demo Application

Show Profile   Welcome **Justin** (Sign out)

Set Current Location: [ ] Go

Query:

SELECT * From Restaurants;

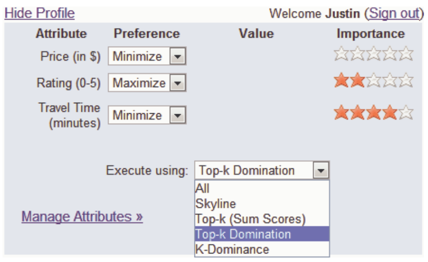
Return top 8 results   Execute

Executed Preference Query:

SELECT * From Restaurant R, Review RV Where R.id=RV.id
PREFERRING R.Price d1, RV.Rating d2, TravelFn(Usr.Loc, MSPRoad_Net, WSTraffic, R.Location) d3
USING SKYLINE Lowest d1, Highest d2, Lowest d3

Copyrights © University of Minnesota, Twin Cities

Figure 3: Client Demo Application



Hide Profile   Welcome **Justin** (Sign out)

Attribute	Preference	Value	Importance
Price (in $)	Minimize		☆☆☆☆☆
Rating (0-5)	Maximize		★★☆☆☆
Travel Time (minutes)	Minimize		★★★★☆

Execute using: Top-k Domination

All
Skyline
Top-k (Sum Scores)
Top-k Domination
K-Dominance

Manage Attributes »

Figure 4: Profile with Preference Method Choice

depicts the application, capable of running on both a web-browser or a mobile device (e.g., Blackberry, iPhone). Users give the application their preference objectives by defining a preference profile for restaurant attributes using a web-based setup screen, as depicted in Figures 4. This screen also allows users to *choose* the preference evaluation method used to answer their query. This choice is mapped to a drop-down box (depicted in Figure 4), that mirrors the available preference methods currently available in the underlying FlexPref DBMS. The user then saves the profile and clicks a button "query", that submits a preference-enhanced SQL query to the underlying FlexPref DBMS. Answers from the DBMS are displayed on the web-based map. Naturally, as users change the preference method in use, they will see different answers returned by the DBMS due to the differing semantics of the preference method.

5. REFERENCES

[1] S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *ICDE*, 2001.

[2] C.-Y. Chan, H. Jagadish, K.-L. Tan, A. K. Tung, and Z. Zhang. Finding k-Dominant Skylines in High Dimensional Space. In *SIGMOD*, 2006.

[3] C.-Y. Chan, H. Jagadish, K.-L. Tan, A. K. Tung, and Z. Zhang. On High Dimensional Skylines. In *EDBT*, 2006.

[4] S. Chaudhuri and L. Gravano. Evaluating Top-K Selection Queries. In *VLDB*, 1999.

[5] J. J. Levandoski, M. F. Mokbel, and M. E. Khalefa. FlexPref: A Framework for Extensible Preference Evaluation in Database Systems. In *ICDE*, 2010.

[6] PostgreSQL: http://www.postgresql.org.

[7] M. L. Yiu and N. Mamoulis. Efficient Processing of Top-k Dominating Queries on Multi-Dimensional Data. In *VLDB*, 2007.