A System for Management and Analysis of Preference Data

Marie Jacob
Univ. of Pennsylvania
majacob@cis.upenn.edu

Benny Kimelfeld
LogicBlox, Inc.
benny.kimelfeld@logicblox.com

Julia Stoyanovich
Drexel University
stoyanovich@drexel.edu

ABSTRACT

Preference data arises in a wide variety of domains. Over the past decade, we have seen a sharp increase in the volume of preference data, in the diversity of applications that use it, and in the richness of preference data analysis methods. Examples of applications include rank aggregation in genomic data analysis, management of votes in elections, and recommendation systems in e-commerce. However, little attention has been paid to the challenges of building a system for preference-data management, which would help incorporate sophisticated analytics into larger applications, support computational abstractions for usability by data scientists, and enable scaling up to modern volumes. This vision paper proposes a management system for preference data that aims to address these challenges. We adopt the relational database model, and propose extensions that are specialized to handling preference data. Specifically, we introduce a special type of a relation that is designed for preference data, and describe composable operators on preference relations that can be embedded in SQL statements, for convenient reuse across applications.

1. INTRODUCTION

Preference data, consisting of orders among sets of items, is a commonplace model to express opinions in a variety of domains. Since the emergence of such datasets, there arose many applications with the need to perform a range of special analytical tasks over them. Consider, for example, preferential voting systems for elections. Statistical studies often look at correlating sociodemographic information of voters with their ranking behavior [10, 18], analysis that is useful to political campaigns. In bioinformatics, prioritized gene lists are a common way to express the output of high-throughput experiments; with many experiments producing different gene rankings, there is often a need to find a consensus ranking that is considered stable or reliable [4, 14]. Lastly, e-commerce applications and social networks that wish to recommend items or products to users have recently tended towards looking at comparison-based feedback rather than at traditional ratings [3, 6]. While much recent work has focused on incorporating preferences over tuples to improve database querying (i.e., *preferences for data management*) [2,12,13,16], we are not aware of any work on general solutions for managing preference data (i.e., *data management for preferences*). Yet, modern volumes, variety and availability of preference data, as well as the interest in sophisticated analytics over such data, naturally motivate challenges involved in general data management. These include appropriate data models, abstractions and system implementations for a large range of applications.

This paper describes our vision of a system for management and analysis of preference data. We propose to adapt the relational database model, and to extend it with a novel kind of a relation, namely *preference relation*. A preference relation can be provided directly as a base relation (alongside ordinary relations), or be derived as a view. Importantly, rather than specifying a single preference order over tuples (as in *p-relations* in [2]), an instance of our preference relation contains a set of *sessions*, each stating a set of pairwise preferences attributable to a particular judge, be it a user or an automatic process. In this way we are able to model divergent opinions, and can accommodate a wide range of preference types (linear rankings, partial orders, inconsistent preferences with cycles, etc.).

In the following sections, we recall common analytics over preference data, propose a formal framework, outline the challenges involved in building primitives for management and analytics (over preference data), and finally explain how our vision complements the state of the art.

2. COMMON OPERATIONS

Various communities in computer science, including machine learning, theoretical CS and database systems, have looked at operations concerning preference data. Here we recall some important operations among those studied.

Rank Aggregation [1, 7–9]. This operation, which has been well studied in theoretical computer science, aggregates a given collection of input rankings to find a ranking that intuitively represents a consensus or summary of all rankings. Aggregation methods range from very simple, such as sorting items on the sum of their ranks (the Borda count), to more sophisticated, where one searches for rankings that minimize the overall distance from each ranking in the input set (Kemeny optimal aggregation).

Clustering over Preferences [5,15,17]. Clustering over preference data partitions the users into a small number

 Articles from this volume were invited to present their results at the 40th International Conference on Very Large Data Bases, September 1st - 5th 2014, Hangzhou, China.
Proceedings of the VLDB Endowment, Vol. 7, No. 12

of groups so that each group consists of users who are in high agreement on the ordering of items. This topic has received more recent attention in machine learning, which focuses on learning statistical distance-based models underlying the data. A challenge in such learning is in utilizing different kinds of training data, such as complete rankings, top-k rankings, partitioned preferences or in the most general case, pairwise preferences.

Sequence Mining [19, 20]. A related area is large-scale sequence mining, where the goal is to find either contiguous or non-contiguous sequences that have high support. Applications include finding n-grams over large text corpora or finding interesting events in time-series data. Although this data weakly corresponds to preferences, the linear ranking properties are similar and thus share motivation in the tasks of querying over large scale ordered data.

Note that some of the above applications use common operations. Learning statistical models of preferences requires computing distances between two preference inputs, as may be required in rank aggregation methods. Likewise, frequencies of partial rankings tends to be useful in both sequence mining and in computing relative frequency distributions for machine learning. Based on this observation, our goal is to create a general system for the management and analysis of preference data, which will provide (1) a unified data model for various types of preferences, including top-k rankings, linear orderings, partitioned preferences, and general pairwise preferences; and (2) a set of primitive functions that is expressive enough for many analytical queries over preferences, and can be computed efficiently at scale.

In the next section, we propose a data model for preferences, and a series of functions that operate over a preference relation. We then briefly describe extensions to existing SQL to support this model, and outline examples of analytical queries that can employ such functions.

3. DATA MODEL

Our data model extends the ordinary relational model with a special type of relation, along with special operations over that relation. Specifically, a *relation schema* $\mathbf{S}$ consists of two types of relation symbols: *ordinary* relation symbols and *preference* relation symbols. An ordinary relation symbol R of a schema $\mathbf{S}$ is associated with a set of *attributes* that we denote by $\mathcal{A}(R)$. A *tuple* t over R is a mapping from $\mathcal{A}(R)$ to atomic values (e.g., numbers, strings, etc.), and a *relation* over R is a finite set of tuples over R.

A preference relation symbol represents a collection of *sessions*, where each session consists of pairwise preferences among *items*. A tuple of a preference relation symbol can be conceptually viewed as a triple (s, l, r) stating that "in session s, item l is preferred to item r." Formally, a preference relation symbol P of a schema $\mathbf{S}$ is associated with three constructs, $\mathcal{S}(P)$, $\lambda(P)$ and $\rho(P)$, where $\mathcal{S}(P)$ is a set of attributes (designated for the session identifier), $\lambda(P)$ and $\rho(P)$ are sequences of attributes (designated for the left-hand-side and the right-hand-side item identifiers, respectively), $\lambda(P)$ and $\rho(P)$ are of the same length, and $\mathcal{S}(P)$, $\lambda(P)$ and $\rho(P)$ are pairwise disjoint. For a preference relation symbol P, we let $\mathcal{A}(P)$ denote the set of all attributes that occur in $\mathcal{S}(P)$, $\lambda(P)$ and $\rho(P)$. A *tuple* and a *relation* over P is defined similarly to an ordinary relation symbol R (replacing $\mathcal{A}(R)$ with $\mathcal{A}(P)$). A relation over a preference relation symbol is called simply a *preference relation*.

A *database instance* D over a schema $\mathbf{S}$ maps every (ordinary and preference) relation symbol Q of $\mathbf{S}$ to a relation, denoted Q^D. When there is no risk of ambiguity, we may abuse the notation and write just Q instead of Q^D.

Note that the above model of preferences naturally corresponds to a directed graph model (each edge corresponds to a tuple in the preference relation). Without any constraints, a preference graph may contain cycles, indicating inconsistencies in preferences. Such inconsistencies are a natural occurrence as the session may correspond to a group of users who conflict with each other in their preferences, or at the individual level, where a user has stated conflicting preferences over time. Reasoning about properties of acyclicity and consistency will be necessary and important operations for analytical queries.

EXAMPLE 3.1. As a shorthand notation, an ordinary relation symbol R is denoted by $R(A_1, \ldots, A_k)$ to specify that $\mathcal{A}(R) = \{A_1, \ldots, A_k\}$, and a preference relation symbol P is denoted by $P(A_1, \ldots, A_k; B_1, \ldots, B_m; C_1, \ldots, C_m)$ to denote that $\mathcal{S}(P) = \{A_1, \ldots, A_k\}$, $\lambda(P) = (B_1, \ldots, B_m)$ and $\rho(P) = (C_1, \ldots, C_m)$. In our running example, the schema $\mathbf{S}$ has 4 relation symbols: Foodies(ssn, age, income), Restaurants(rid, cuisine, price), Surveys(sid, ssn), and the relation Opinions(sid; rid1; rid2) Note that Persons, Restaurants and Surveys are ordinary relations, while Opinions is a preference relation, stating user preferences of restaurants. □

A *preference-to-preference* operator is a function that maps a preference relation P to another preference relation Q, such that $\lambda(Q) = \lambda(P)$ and $\rho(Q) = \rho(P)$. Below we list various useful tasks that can be viewed as such operations. (Those are just examples, and the list will naturally extend as we materialize our vision.)

Transitive closure. When applied to a preference relation P, this relation contains a triple (s, l, r) whenever there is a (nonempty) sequence of triples (s, l_i, r_i) that form a path from l to r.

Selection. An operator of this sort returns a subset of the input relation P, based on a condition on the sessions and/or the individual preferences. Such a condition can be "the session is a linear/acyclic order," or "the preference does not participate in any cycle within the session."

Rank aggregation. The result of applying this operator to the relation P is a preference relation Q with $\mathcal{S}(Q) = \emptyset$. The relation Q consists of a single session that is meant to be a representative of the preferences of all the sessions. This operator also has a *grouped* version that is parametrized by a subset S' of $\mathcal{S}(P)$; in that case, $\mathcal{S}(Q) = S'$ and the aggregation is applied to each distinct value of S' separately.

Clustering. This operator is parametrized by the number N of desired clusters. The result for the relation P is a preference relation Q with $\mathcal{S}(Q) = \mathcal{S}(P) \cup \{\mathsf{cluster}\}$, where cluster is a distinguished attribute. The relation Q is the same as P, except that the attribute cluster denotes the identifier of the cluster to which the session belongs.

Sequence/graph mining. This operator is parametrized by a frequency τ; the result is a preference relation Q with $\mathcal{S}(Q) = \{\mathsf{id}\}$, consisting of (maximal) preferences with a support of (i.e., occur orderly in path of) at least τ sessions of P. Like in the case of rank aggregation, there is also a grouped version parametrized on a subset S' of $\mathcal{S}(P)$, and in that case $\mathcal{S}(Q) = S' \cup \{\mathsf{id}\}$.

4. SQL EXTENSION

We now present a proposed extension of SQL that supports the management and analysis of preference data.

The first extension is in the generation of new preference relations. When we create a preference relation P, we need to specify $\mathcal{S}(P)$, $\lambda(P)$ and $\rho(P)$. We do so by using the keyword SELPREF (i.e., select preference), and by using semicolons to distinguish between the three chunks of attributes.

$$\text{SELPREF } A_1, \ldots, A_k \text{ ; } B_1, \ldots, B_m \text{ ; } C_1, \ldots, C_m$$

As usual, each A_i, B_i and C_i is an attribute name from the relations mentioned in the FROM clause of the query. Also as usual, one can use $R.A$ if the attribute A occurs in more than one FROM relation, and an attribute A can be renamed using the AS keyword.

EXAMPLE 4.1. We continue our running example. The following query constructs a preference relation, where opinions of a foodie are collected from all her surveys.

```
SELPREF S.ssn ; O.rid1 ; O.rid2
   FROM Opinions O, Surveys S
  WHERE S.sid = O.sid
```

Here, the preference relation P is constructed with $\mathcal{S}(P) = \{\text{ssn}\}$, $\lambda(P) = \{\text{rid1}\}$, and $\rho(P) = \{\text{rid2}\}$. □

The next extension is by preference-to-preference operators, which can be used in the FROM clause instead of base relations. The general syntax for such an operator is: $X[P]$, where X is the name of the function (e.g., a rank-aggregation algorithm) and P is a preference relation symbol. An example would be TransClosure[Opinions], where TransClosure is the name assigned to the transitive-closure function. Expected parameters for the function (e.g., N and τ) are put in angles after X, and grouping can be specified through the addition of BY $(A_1, \ldots, A_k)$, where $A_1, \ldots, A_k$ are in $\mathcal{S}(P)$. For example, Cluster⟨10⟩[OpinionsByAge] BY (age) clusters the opinions of the foodies in each age group into 10 clusters. As is conventional in SQL, one can use a nested query instead of the preference relation symbol P.

EXAMPLE 4.2. We continue our running example. The following query selects all demographic groups of foodies who prefer "Momofuku" to "Bozu" and "Bozu" to "Qi" (Asian Fusion restaurants in NYC).

```
CREATE VIEW ExtOpinions AS
  SELECT age, income, rid1, rid2
    FROM TransClosure[ RankAgg[
      SELPREF F.ssn, age, income ; rid1 ; rid2
         FROM Opinions O, Surveys S, Foodies F
        WHERE S.sid = O.sid AND S.ssn = F.ssn
     ] BY (age,income) ]

SELECT O1.age, O1.income
  FROM ExtOpinions O1, ExtOpinions O2
 WHERE O1.age = O2.age AND O1.income = O2.income
   AND O1.rid1 = 'Momofuku' AND O1.rid2 = 'Bozu'
   AND O2.rid1 = O1.rid2 AND O2.rid2 = 'Qi'
```

Note that the view definition computes an aggregated preference for each group of people as defined according to their age and income: combining the preference relation `Opinions` with `Surveys` and `Foodies` results in a new preference relation with session identifiers corresponding to `ssn, age, income`. Computing the transitive closure of this resulting relation materializes direct links (preferences) between restaurants. Using this view, we can compute the desired result by performing a self-join to reconstruct two-hop paths. □

EXAMPLE 4.3. Consider a query to find restaurants that are most preferred by users. We can define this to be the restaurants that occur in at least 1000 sessions, and are preferred to at least 5 other restaurants.

```
SELECT rid1
  FROM TransClosure[MaxFrequent<1000>[Opinions]]
 GROUP BY rid1
HAVING COUNT(*) > 5
```

This query finds all maximal frequent preferences over the `Opinions` table, which results in a new preference relation P. This relation consists of preferences that occur in at least 1000 surveys (i.e. substructures that are common among at least 1000 preference graphs). To find restaurants that are preferred to at least 5 others, we take the transitive closure of P, which gives us direct pairwise preferences. A grouping and selection over the first restaurant (rid1) eliminates restaurants that are preferred to fewer than 5 others. □

5. OUTLOOK

There are a number of significant challenges in materializing our proposed vision for efficient management and analysis of preference data. We are in the process of implementing the necessary facilities as part of PostgreSQL, which has strong support for extensibility features. Nonetheless, significant advances are required to make working with preference relations practical. Specifically, we must study semantic aspects such as integrity constraints and algebraic properties of preference-to-preference functions, develop efficient physical representations and access methods for preference relations, and implement scalable analytics. In what follows, we discuss these challenges and our research plans towards solutions.

Semantic Aspects. Applications that manage preference data may involve integrity constraints; for example, some preference relations need to be partial/linear orders. We would like to enable users to easily specify such constraints. Importantly, such constraints may be used for static analysis of queries, and in particular, for inferring properties of preference relations generated from base preferences. In addition, we would like to determine whether a query is sensitive to key invariants of the input preferences (e.g., the query has the same results on base preference relations with the same transitive closure). Such inferred knowledge is useful not only to guarantee the consistency of data, but also for optimization by query rewriting or selection of indexes.

Physical Representation. To illustrate an aspect of representation, consider a preference relation P, and suppose that preferences in an instance of P correspond to a total linear ordering of the items. In this case, it is beneficial to create an alternative physical representation that stores P using numerical ranks rather than pairwise comparisons. Using this representation, we can compute the transitive closure of preferences with a simple self-join, rather than having to execute a recursive query on the pairwise representation.

While it is clear that an alternative physical representation of this kind is useful for linear orders, we must view this

in the context of query optimization and rewriting, evaluating opportunities to incorporate operator reordering, lazy evaluation and other optimization techniques, towards more efficient query execution plans. This will require an understanding of, and the ability to reason about, the algebraic properties of queries extended with preference-to-preference functions. For example, we will need to determine whether a linear ordering that holds in an instance of P is preserved as P is manipulated by relational operations and by preference analytics. Other special cases that are likely to warrant customized physical representations include partial orders, where operations like the transitive closure are well-motivated, and preferences with disconnected graphs.

Scalable Analytics. Another important challenge is in translating preference-to-preference functions, and more generally queries that embed such functions, into efficient executions. We need to scale up these operations to large volumes of preference data, where the volume is due to either the number of sessions, or the number of items, or both. Facing this challenge may involve compromises on the guarantees of the algorithms, to allow for practical execution costs. For instance, consider rank aggregation. Several rank aggregation methods have been discussed in the literature and would be useful to implement, such as Kemeny optimal aggregation [11]. This rank aggregation method is particularly desirable because it adheres to the Extended Condorcet Criterion (ECC) [23], stating that if a majority of the rankers prefer i to j then the aggregate ranking should prefer i to j. Kemeny aggregation is known to be NP-hard [7], and several attempts have been made to provide tractable relaxations [1, 7, 22]. Subbian and Melville [22] essentially propose a *quick sort* on elements based on majority precedence, while Dwork et al. [7] propose *local Kemenization*, which corresponds to bubble sort. It would be interesting to understand the applicability of these algorithms to various settings where rank aggregation is required, and to evaluate the scalability and quality of the resulting solution. A promising avenue of research is to evaluate performance optimization opportunities that result from exploiting parallelism available on modern hardware.

6. SYNERGY WITH PRIOR ART

The work proposed here is motivated by and aligns with our long-term project that focuses on understanding local structure in ranked datasets [21].

Recent trends in making database management systems more *preference-aware*, have been focusing on specific types of preference queries over relational data. The work of Arvanitis and Koutrika [2], for instance, looks at optimizing queries for the most preferred answers, given a set of user-declared preferences. RankSQL [16] proposes new rank relations and algebraic operators as extensions to an existing DBMS to optimize top-k queries. The common thread that ties many of these systems together is the idea of finding a *single* preference over a relation or a set of query results, given high-level preference functions or formulae. In contrast, no system has yet looked at general queries over *collections* of individual preferences.

In a seminal line of work, Kießling et al. [12, 13] propose mechanisms to generalize ranking in query results beyond the traditional "ORDER BY" and "LIMIT" of SQL. These mechanisms include functions that construct partial orders from columns, and a collection of algebraic operators to compose partial orders into new ones. These partial orders are constructed and used as part of relational queries, and are not treated as ordinary data. For that reason, their basic conceptual construct is a partial order over an ordinary relation, while our preference relations store a *collection* of (or *sessions* of) preferences. Importantly, we believe that the two research directions can be naturally combined towards a richer formalism. Analytics can be used to synthesize preferences representative of whole populations, which can in turn be used to enrich the set of atomic partial orders of Kießling et al. [12, 13].

7. REFERENCES

[1] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: ranking and clustering. *Journal of the ACM (JACM)*, 55(5):23, 2008.

[2] A. Arvanitis and G. Koutrika. Towards preference-aware relational databases. In *ICDE*, pages 426–437, 2012.

[3] S. Balakrishnan and S. Chopra. Two of a kind or the ratings game? adaptive pairwise preferences and latent factor models. *Frontiers of Computer Science*, 6(2):197–208, 2012.

[4] A.-L. Boulesteix and M. Slawski. Stability and aggregation of ranked gene lists. *Briefings in Bioinformatics*, 10(5):556–568, 2009.

[5] L. M. Busse, P. Orbanz, and J. M. Buhmann. Cluster analysis of heterogeneous rank data. In *ICML*, pages 113–120. ACM, 2007.

[6] A. Das Sarma, A. Das Sarma, S. Gollapudi, and R. Panigrahy. Ranking mechanisms in twitter-like forums. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, pages 21–30, New York, NY, USA, 2010. ACM.

[7] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW*, pages 613–622. ACM, 2001.

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

[9] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66(4):614–656, 2003.

[10] I. C. Gormley and T. B. Murphy. A mixture of experts model for rank data with applications in election studies. *The Annals of Applied Statistics*, 2(4):1452–1477, 12 2008.

[11] J. Kemeny. Mathematics without numbers. *Daedalus*, 88, 1959.

[12] W. Kießling. Foundations of preferences in database systems. In *VLDB*, pages 311–322. Morgan Kaufmann, 2002.

[13] W. Kießling, M. Endres, and F. Wenzel. The preference sql system - an overview. *IEEE Data Eng. Bull.*, 34(2):11–18, 2011.

[14] R. Kolde, S. Laur, P. Adler, and J. Vilo. Robust rank aggregation for gene list integration and meta-analysis. *Bioinformatics*, 28(4):573–580, 2012.

[15] G. Lebanon and Y. Mao. Non-parametric modeling of partially ranked data. In *NIPS*, volume 7, pages 857–864, 2007.

[16] C. Li, K. C.-C. Chang, I. F. Ilyas, and S. Song. RankSQL: Query algebra and optimization for relational top-k queries. In *SIGMOD Conference*, pages 131–142, 2005.

[17] T. Lu and C. Boutilier. Learning mallows models with pairwise preferences. In *ICML*, pages 145–152, 2011.

[18] G. McElroy and M. Marsh. Candidate gender and voter choice: Analysis from a multimember preferential voting system. *Political Research Quarterly*, 63(4):pp. 822–833, 2010.

[19] I. Miliaraki, K. Berberich, R. Gemulla, and S. Zoupanos. Mind the gap: Large-scale frequent sequence mining. In *SIGMOD*, pages 797–808, New York, NY, USA, 2013. ACM.

[20] S. Parthasarathy, M. J. Zaki, M. Ogihara, and S. Dwarkadas. Incremental and interactive sequence mining. In *CIKM*, pages 251–258. ACM, 1999.

[21] J. Stoyanovich, S. Amer-Yahia, S. B. Davidson, M. Jacob, and T. Milo. Understanding local structure in ranked datasets. In *CIDR*, 2013.

[22] K. Subbian and P. Melville. Supervised rank aggregation for predicting influence in networks. *CoRR*, abs/1108.4801, 2011.

[23] M. Truchon. An extension of the Condorcet criterion and Kemeny orders. *J. Eco. Lit.*, 1998.