Towards a logical framework for OLAP query log manipulation

Julien Aligon, Dominique Li, Patrick Marcel, Arnaud Soulet
Université de Tours
LI lab., France
Firstname.Lastname@univ-tours.fr

ABSTRACT
This paper proposes a manipulation language tailored for OLAP query logs, stemming from the relational algebra. This language is based on binary relations over sequences of queries (called sessions). We propose two such relations allowing to group and order sessions. Examples of expressions in this language illustrate its interest for various user-centric approaches, like query recommendation or log summarization, particularly relevant in the context of data warehouse exploration.

1. INTRODUCTION
It has recently been pointed out the necessity to come up with flexible, powerful means for analyzing past issued queries, usually stored in the DB query log, and decompose, store and handle them in a dedicated subsystem in order to better support any decisional task with the knowledge captured in these queries [9]. As examples of such tasks, the system should be able to support the user when formulating new queries and leverage his / her knowledge with other users (query recommendation) [10, 5]. Tests with users demonstrated that browsing through past SQL query sessions indeed helped speed up query composition [11].

Achieving such assistance is particularly relevant in the context of multidimensional databases, where sequences of queries (called analytical sessions, or simply sessions) are used to navigate data warehouses and identify regions of anomalies that may represent problem areas or new opportunities [16]. More precisely, [15] reported a study of a query log of 18 users over a two months period in a large chemical company. The log showed that users interactively formulate their next query based on the result of the previous query, illustrating the navigational nature of an analytical session over a data cube. OLAP query logs can thus be used for learning user preferences for query personalization [3] or for query recommendation [6, 7].

In both contexts, query logs can be seen as sets of sequences of queries. While query languages for manipulating sequences have been proposed (see e.g., [12]), to the best of our knowledge, no logical framework exist for log management aiming at user empowerment. A somewhat similar effort to ours is the REQUEST language [1], which is dedicated to recommendations, and does not give a first class citizenship to logs.

2. RELATED WORK
While many RDBMs provide query logging, logs are used essentially for physical tuning. Leveraging query logs for user empowerment has recently attracted attention in the domain of relational databases [9]. Logs can indeed be used for learning user preferences [8], for query recommendation [5, 17], for query auto-completion [10], or for query composition [11].

Similar approaches can also be found in the particular context of multidimensional databases, where sequences of queries (called analytical sessions, or simply sessions) are used to navigate data warehouses and identify regions of anomalies that may represent problem areas or new opportunities [16]. More precisely, [15] reported a study of a query log of 18 users over a two months period in a large chemical company. The log showed that users interactively formulate their next query based on the result of the previous query, illustrating the navigational nature of an analytical session over a data cube. OLAP query logs can thus be used for learning user preferences for query personalization [3] or for query recommendation [6, 7].

In both contexts, query logs can be seen as sets of sequences of queries. While query languages for manipulating sequences have been proposed (see e.g., [12]), to the best of our knowledge, no logical framework exist for log management aiming at user empowerment. A somewhat similar effort to ours is the REQUEST language [1], which is dedicated to recommendations, and does not give a first class citizenship to logs.

3. PRELIMINARY DEFINITIONS
We introduce the basic definitions underlying the framework.

3.1 Multidimensional schema and queries
We start with the definitions of a multidimensional schema and OLAP queries over it.
Definition 3.1. A multidimensional schema (or, briefly, a schema) is a triple $\mathcal{M} = (A, H, \text{Meas})$ where:

- $A$ is a finite set of levels (categorical attributes), whose domains are assumed pairwise disjoint,
- $H = \{h_1, \ldots, h_n\}$ is a finite set of hierarchies, a hierarchy $h_i$ being a set $\text{Lev}(h_i) = \{L_0, \ldots, L_k\}$ of levels together with a roll-up total order $\succeq_{h_i}$ of $\text{Lev}(h_i)$.
- $\text{Meas}$ is a finite set of measure attributes, each defined on a numerical domain $\text{Dom}(m)$.

For each hierarchy $h_i$, the bottom level of the order, noted $\text{DIM}_m$, determines the coarsest aggregation level for the hierarchy. Conversely, the top level, noted $\text{ALL}$, has a single possible value and determines the coarsest aggregation level.

A group-by set includes one level for each hierarchy, and defines a possible way to aggregate data. A reference (or coordinate) of a group-by set is a point in the $n$-dimensional space defined by the levels in that group-by set.

Definition 3.2. Given schema $\mathcal{M} = (A, H, \text{Meas})$, let $\text{Dom}(H) = \text{Lev}(h_1) \times \cdots \times \text{Lev}(h_n)$; each $G \in \text{Dom}(H)$ is called a group-by set of $\mathcal{M}$. Let $G = \{a_{1i}, \ldots, a_{ni}\}$ and $\text{Dom}(G) = \text{Dom}(a_{1i}) \times \cdots \times \text{Dom}(a_{ni})$; each $g \in \text{Dom}(G)$ is called a reference (or a coordinate) of $G$.

Let $\succeq_H$ denote the product order$^1$ of the roll-up orders of the hierarchies in $H$. Then, $(\text{Dom}(H), \succeq_H)$ is a lattice, that we will call group-by lattice, whose top and bottom elements are $G^\perp = (\text{DIM}_1, \ldots, \text{DIM}_n)$ and $G^\top = (\text{ALL}, \ldots, \text{ALL})$, respectively.

Example 3.1. We introduce a running example. IPUMS is a public database storing census microdata for social and economic research [13]. Its CENSUS multidimensional schema includes the five hierarchies whose roll-up orders are shown in Figure 1, and measures AvgIncome, AvgCostGas, AvgCostWatr, and AvgCostElect.

More formally, its schema is $\text{CENSUS} = (\text{A}_{\text{CENSUS}}, H_{\text{CENSUS}}, \text{M}_{\text{CENSUS}})$ with:

- $\text{A}_{\text{CENSUS}} = \{\text{City, State, Region, AllCities, Race, RaceGroup, MRN, AllRaces, Year, AllYears, Sex, AllSexes, Occ, AllOccs}\}$
- $H_{\text{CENSUS}} = \{\text{RESIDENCE, RACE, TIME, SEX, OCCUPATION}\}$
- $\text{M}_{\text{CENSUS}} = \{\text{AvgIncome, AvgCostGas, AvgCostWatr, AvgCostElect}\}$

For instance, hierarchy $\text{RESIDENCE}$ is the set of levels $(\text{City, State, Region, AllCities})$ with $\text{AllCities} \succeq_{\text{RESIDENCE}} \text{Region} \succeq_{\text{RESIDENCE}} \text{State} \succeq_{\text{RESIDENCE}} \text{City}$.

Examples of group-by sets are:

- $G_0 = G^\perp = (\text{City, Race, Year, Occ, Sex})$
- $G_1 = (\text{Region, Mrn, Year, Occ, Sex})$
- $G_2 = G^\top = (\text{AllCities, AllRaces, AllYears, AllOccs, AllSexes})$

An OLAP query is modeled by a triple structuring the 3 components of a multidimensional query: A group-by set, a set of selection predicates and a measure set.

Definition 3.3. The model of a query over schema $\mathcal{M} = (L, H, \text{Meas})$ is a triple $q = (G, P, M)$ where:

1The product order of $n$ total orders is a partial order on the Cartesian product of the $n$ totally ordered sets, such that $(x_1, \ldots, x_n) \succeq (y_1, \ldots, y_n)$ if $x_i \succeq y_i$ for $i = 1, \ldots, n$.

\[ \begin{array}{c|c|c|c|c|c|c|c|c|c|c|c|c|c|c} \text{City} & \text{Race} & \text{State} & \text{RaceGroup} & \text{Region} & \text{Min} & \text{Year} & \text{Sex} & \text{Occ} \\
\hline \text{AllCities} & \text{AllRaces} & \text{AllYears} & \text{AllSexes} & \text{AllOccs} \\
\end{array} \]

Figure 1: Roll-up orders for the five hierarchies in the CENSUS schema (Mrn stands for MajorRaceNumber).

1. $G \in \text{Dom}(H)$ is the query group-by set;
2. $P = \{p_1, \ldots, p_n\}$ is a set of Boolean predicates, one for each hierarchy, whose conjunction defines the selection predicate for $q$; they are of the form $l = v$, or $l \notin V$, with $l$ a level, $v$ a value, $V$ a set of values. Conventionally, we note $p_i = \text{TRUE}$ if no selection on $h_i$ is made in $q$ (all values being selected);
3. $M \subseteq \mathcal{M}$ is the measure set whose values are returned by $q$.

Example 3.2. An example of model of a query on the CENSUS schema is: $q_1 = \langle G^\top, \text{TRUE, AvgIncome} \rangle$ where $\text{TRUE} = \{\text{TRUE}_{\text{RESIDENCE}}, \ldots, \text{TRUE}_{\text{SEX}}\}$.

3.2 Specialization relation over queries

Note that the specialization relations we considered throughout the paper are partial orders [14].

Given a schema $(L, H, \text{Meas})$, the group-by sets, selection predicates and sets of measures of the queries that can be expressed over this schema, can all be respectively arranged into a lattice. The operations for manipulating unvaluated queries use this property.

Considering two group by sets $G$ and $G'$, recall that we note $G \succeq_H G'$ if $G$ is more general$^2$ then $G'$ in the sense of the group-by lattice. We define the $\text{supp}$ and $\text{inf}$ operations on group-by sets, to be, $\text{supp}(G, G') = \text{min}_{g \in H} \{g'' \in \text{Dom}(H) : g'' \succeq_H G \land G'' \succeq_H G'\}$ and $\text{inf}(G, G') = \text{max}_{g \in H} \{g'' \in \text{Dom}(H) : g'' \succeq_H G \land G'' \succeq_H G'\}$.

For two sets of selection predicates $P$, $P'$, we note $P \supseteq_P P'$ if $P$ is more selective than $P'$, i.e., $\text{val}(P) \subseteq \text{val}(P')$, where $\text{val}(X)$ is the set of values used in predicate $X$. Note that, whatever $p_i \in P$, it is $\text{val}(p_i) \subseteq \{\text{TRUE}\}$, $\text{val}(p_i) \cap \{\text{TRUE}\} = \{\text{val}(p_i)\}$ and $\text{val}(p_i) \cup \{\text{TRUE}\} = \{\text{TRUE}\}$.

We define the $\text{supp}$ and $\text{inf}$ operations on sets of predicates, to be, $\text{supp}(P, P') = \text{val}(P) \cap \text{val}(P')$ and $\text{inf}(P, P') = \text{val}(P) \cup \text{val}(P')$. We recall that predicates in $P$ (respectively $P'$) are interpreted as a conjunction of Boolean predicates that indicate the values selected. Therefore, a query $(G, P, M)$ where $P$ features no selection predicate for a hierarchy $h_i$, is interpreted as a query selecting nothing, i.e., the unsatisfiable query. We thus define $P^\top$ as the set of

\[ \text{Note that } G \text{ can also be interpreted as being more selective as } G', \text{ in the sense that it asks for less values of the hierarchies to be retrieved.} \]
sets of predicates where there exists one such hierarchy, i.e., $P^\top = \{P\}$ is a set of selection predicates for a schema $(L,H,Meas)$ with $n$ hierarchies, and $|P| < n$.

For two measure sets $M, M'$, we note $M \succeq_m M'$ if $M \subseteq M'$. We define the $\text{supp}$ and $\text{inf}$ operations on measure sets, to be, $\text{supp}(M, M') = M \cap M'$ and $\text{inf}(M, M') = M \cup M'$.

Given a schema $(L,H,Meas)$, we consider the set of queries over this schema that are the less informative ones, that we note $Q^\top = \{(G, P, M)|G = G^\top \vee P \in P^\top \vee M = \emptyset\}$. Among the queries in that set, a particular one is $q^\top = (G^\top, \emptyset, \emptyset)$. Note that by less informative, we mean that its expression may not reflect a user’s intention. For instance, a query $(G, P, \emptyset)$ is interpreted as having no measure, thus nothing can be said about the user’s intention in terms of measures to be analyzed.

We can now define the specialization relation over queries.

**Definition 3.4.** Let $q = (G, P, M)$ and $q' = (G', P', M')$ be two queries over the same schema. $q$ is more general than $q'$, noted $q \succeq q'$, if $G \supseteq G'$ and $P \supseteq P'$ and $M \succeq_m M'$.

Given a schema $(L,H,Meas)$, the set of queries over this schema together with the specialization relation $\succeq$ over queries form a lattice (being the product of three lattices).

In particular, let $q = (G, P, M)$ and $q' = (G', P', M')$ be two queries. Their most specific common ancestor is $\text{anc}(q, q') = (\text{supp}_G(G', G), \text{supp}_P(P', P), \text{supp}_M(M, M'))$.

**Example 3.3.** Consider the following queries over the schema CENSUS schema of Example 3.1:

$q_1 = \{(\text{City, AllRaces, Year, AllOccs, AllSexes}) \},
\{\text{Region} \in \{\text{Pacific, Atlantic}\}, \text{Year} = "2001", \text{TRUE\textunderscore RACE, TRUE\textunderscore OCCUPATION, TRUE\textunderscore SEX},
\text{AvgIncome}\}$

$q_2 = \{(\text{Region, AllRaces, Year, AllOccs, Sex}) \},
\{\text{Sex} \in \{\text{Male, Female}\}, \text{Year} = "2001", \text{TRUE\textunderscore RACE, TRUE\textunderscore RESIDENCE, TRUE\textunderscore OCCUPATION},
\text{AvgIncome}\}$

Their most specific common ancestor with respect to $\succeq$ is

$q_3 = \{(\text{Region, AllRaces, Year, AllOccs, AllSexes}) \},
\{\text{Year} = "2001", \text{TRUE\textunderscore RACE, Sex} \in \{\text{Male, Female}\}
\text{Region} \in \{\text{Pacific, Atlantic}\}, \text{TRUE\textunderscore OCCUPATION},
\text{AvgIncome}\}$

Note that, had $q_2$’s selection predicate included Year = "2002" instead of Year = "2001", then the most specific common ancestor would have been in $Q^\top$.

### 3.3 Sessions and logs

In an OLAP context, queries are usually not isolated from one another. We thus define an analytical session, or session for short, as a sequence of queries.

**Definition 3.5.** Let $S$ be a set of queries over schema $M = (A,H,M)$. A session $s$ of $k$ queries $s = \langle q_1, \ldots, q_k \rangle$ over $M$ is a function from an ordered set $\text{pos}(s)$ of integers (called positions) of size $k$ to $S$.

A log $L$ is a finite set of sessions, noted $L = \{s_1, \ldots, s_p\}$.

**Definition 3.6.** Let $M = (A,H,M)$ be a schema. A log $L$ is a finite set of sessions over $M$.

We denote the set of queries of a session $s$ by $\text{queries}(s)$ and the set of queries of a log $L$ by $\text{queries}(L)$. We note $q \in L$ for a log $L$ if $q \in \text{queries}(L)$.

**Example 3.4.** An example of OLAP session of length 3 on the CENSUS schema is $s = \langle q_1, q_2, q_3 \rangle$, where the query models are:

$q_1 = (G^\top, \text{TRUE}, \text{AvgIncome})$
$q_2 = (G_1, \text{TRUE}, \text{AvgIncome})$
$q_3 = (G_1, \{\text{Year} = 2011, \text{TRUE\textunderscore RACE}, \ldots\}, \text{AvgIncome})$

where TRUE = \{\text{TRUE\textunderscore RESIDENCE, TRUE\textunderscore RACE, TRUE\textunderscore TIME, TRUE\textunderscore OCCUPATION, TRUE\textunderscore SEX}\}. Note that the user here applied a roll-up operator to move from $q_1$ to $q_2$, and a slice operator to move from $q_2$ to $q_3$.

### 4. Binary Relations over Sessions

We introduce two binary relations over sessions: a specialization relation, aiming at ordering sessions, and a similarity relation, aiming at grouping sessions.

#### 4.1 Specialization Relation over Sessions

We recall that a session is a sequence of queries. In particular, we note $S^\top$ the set of sessions containing a query in $Q^\top$.

We introduce a specialization relation over sessions that is based on a specialization relation over queries. In what follows, $s \succeq s'$ denotes that a session $s$ is more general than a session $s'$. Note that the relations over sessions are defined based on a relation $\succeq$ over queries whose semantics is not fixed. For instance, interpreting $\succeq$ as a preference relation over queries in what follows, allows to define preference relations over sessions.

**Definition 4.1.** A session $s = \langle q_1, \ldots, q_n \rangle$ is more general than another session $s' = \langle q_1', \ldots, q_n' \rangle$, written $s \succeq s'$, if there exists a sequence of $n'$ integers $\{i_1, \ldots, i_{n'}\}$ with $i_1 = 1$, $i_{n'} = n$ and, for $k \in \{1, \ldots, n' - 1\}$, $i_k \leq i_{k+1}$, such that, for all $j \in \{1, \ldots, n\}$, $i_j$ is $q_j \preceq q_j'$.

The intuition of this specialization relation is given Figure 2. Note that if $s \succeq s'$ then the sequence of integers $\{i_1, \ldots, i_{n'}\}$ defines a partition $P = \{p_1, \ldots, p_{n'}\}$ of $\text{queries}(s')$ where the $p_i$’s can be ordered according to the queries of $s$, and the queries in each $p_i$, each less general than $q_i$, constitute a sub-session of $s'$. Given an integer $n$ and a session $s$ with $|s| \geq n$, we call a $n$-partition of $s$, a partition $P = \{p_1, \ldots, p_n\}$ of $\text{queries}(s)$ such that, for all $i \in \{1, n - 1\}$, all queries of $p_i$ precede the queries in $p_{i+1}$ in $s$.

**Example 4.1.** Consider queries $q_1, q_2, q_3$ of Example 3.3 and sessions $s_1 = \langle q_1, q_2, q_1 \rangle$, $s_2 = \langle q_1, q_1 \rangle$ and $s_3 = \langle q_1 \rangle$. Then we have $s_3 \succeq s_2 \succeq s_1$.

It can easily be seen that $\succeq$ is a specialization relation, being a partial order. However, note that with this specialization relation, sessions cannot be arranged into a lattice. In particular, there is more than one more specific common ancestor to a pair of sessions, as illustrated by the following example.
queries to their measure sets. Consider sessions \( s \) having two more specific ancestors: \( s \) and \( c \). These sessions have two more specific ancestors: \( s'' = \{a,c\}, \{b\}, s''' = \{a\}, \{b\} \).

Finding the most specific ancestors to a pair of sessions \( s, s' \) can be done as follows. First, note that the most specific ancestors of \( s, s' \), assuming \( s \) the shortest, have the length of \( s \), noted \(|s|\). The most specific ancestors can thus be obtained by computing all the \(|s|\)-partitions of \( s' \) and combining each such partition with \( s \). More formally, let \( s = q_1, \ldots, q_n \) a session and \( s' = q_1, \ldots, q_m \) another session such that \(|s| \leq |s'|\). The most specific ancestors to \( s \) and \( s' \) are \( \text{anc}(s, s') = \{(\text{supp}(q_1, p_1), \ldots, \text{supp}(q_n, p_n))\mid \{p_1, \ldots, p_n\} \text{ is a } n\text{-partition of } s'\} \). Note that, the cardinality of this set is at most \( \binom{|s'|}{|s| - 1} \), which is the number of \(|s|\)-partitions of \( s' \). Note that, from this set, we may want to remove the sessions in \( S^\top \).

4.2 Similarity relation over sessions

Measuring session similarity is the focus of an ongoing joint work with University of Bologna [4]. In particular, a similarity measure tailored for OLAP queries is proposed, and classical similarity measures for sequences are extended to match the OLAP context.

Suppose a function \( \text{dist} \) applied to pairs of sessions, giving the distance between two sessions. Binary similarity relations over sessions can be defined from such a function. For instance, \( \text{sim}(s, s') \equiv \text{dist}(s, s') \leq \alpha \) for some real \( \alpha \) indicates that two sessions \( s \) and \( s' \) are similar if their distance is below a threshold.

As another example, given a set \( S \) of sessions, and for some threshold \( \alpha \), \( \text{sim}(s_0, s_n) \equiv \exists s_1, \ldots, s_{n-1} \in S, \forall i \in \{0, \ldots, n - 1\}, \text{dist}(s_i, s_{i+1}) \leq \alpha \) indicates that two sessions \( s_0 \) and \( s_n \) are similar if there exists a chain of sessions with distance under \( \alpha \) that can connect the two sessions. In that case, this relation is an equivalence relation over \( S \).

5. THE LOG MANIPULATION LANGUAGE

We now present the log manipulation language.

5.1 Intuitions

The language introduced below enables to manipulate logs as defined above, i.e., sets of sessions. It is essentially an adaptation of the relation algebra with one main difference. The (extended) relational algebra enables the manipulation of relations, i.e., sets of tuples, that have a schema. The language introduced here manipulates logs, i.e., sets of sessions, that do not have a schema. A schema can therefore not be used in the expressions formed with this language. Instead, the operations of the language are parametrized by a binary relation over sessions \( \theta \), that may for instance be used to compare, order or group sessions. In particular, \( \theta \) can be one of the relations presented above.

5.2 Formal definitions

For the sake of simplicity, the names and symbols of the operations are the same as the ones used in the relational algebra. The language features 5 operations. Two of them are unary operations: selection \( \sigma \) and group by and aggregation \( \pi \). Three of them are binary: join \( \Join \), union \( \cup \) and difference \( \setminus \). We now introduce the formal definitions, that we illustrate with simple examples. In what follows, we consider a relation \( \text{sim} \) that relates two sessions if their distance is under a given threshold, or if they can be connected through a chain of sessions as defined in Section 4.2. Advanced manipulations are given in Section 6.

5.2.1 Selection

The selection operation enables to select from a log those sessions satisfying a given condition. This condition is given under the form of a relation to another session.

**Definition 5.1.** Let \( L \) be a log, \( s \) be a session and \( \theta \) be a binary relation over sessions.

\( \sigma_{\theta,s}(L) = \{s' \in L \mid \theta(s, s')\} \)

**Example 5.1.** The log can be searched for sessions similar to session \( s \), using the sim relation, with expression: \( \sigma_{\text{sim},s}(L) \). This expression is called \( \text{findSimilar}(s, \text{sim}, L) \) in what follows.

The log can be searched for sessions containing a particular query \( q \) with the expression: \( \sigma_{\text{in},(q)}(L) \) where \( \text{in}((q), s') \) is true if \( q \in \text{queries}(s') \). In particular, this expression can be used to find sessions in \( S^\top \) if \( q = q^\top \). This expression is called \( \text{find}S^\top(L) \) in what follows.

5.2.2 Set operations

As logs are defined as sets of sessions, the set union and set difference operations can be used to manipulate logs. The definitions are straightforward:

**Definition 5.2.** Let \( L \) and \( L' \) be two logs.

\( L \cup L' = \{s \in L \text{ or } s \in L'\} \) and \( L \setminus L' = \{s \in L \mid s \notin L'\} \).

**Example 5.2.** Using the \( \geq \) specialization relation over sessions, a log \( L \) can be searched for all sessions that are more specific than another session \( s \), with expression: \( L \setminus \sigma_{\geq,s}(L) \).
5.2.3 Join

The join operation enables to combine the sessions in two logs that are related through a relation \( \theta \).

**Definition 5.3.** Let \( L \) and \( L' \) be logs, \( f \) be a binary functions outputting a session and \( \theta \) be a binary relation over sessions,

\[
L \Join \theta, f, L' = \{ f(s, s') | s \in L, s' \in L', \theta(s, s') \}.
\]

**Example 5.3.** Two logs can be compared using the join operation. For instance, given two logs \( L \) and \( L' \), the sessions that are similar among the logs can be found with the expression:

\[
L \Join_{\text{sim, first}} L' \cup L' \Join_{\text{sim, first}} L \text{ where first}(s, s') = s \text{ for any pair of sessions } s, s'.
\]

A log \( L \) can be searched for the best sessions it contains with respect to a partial order \( \succeq \), i.e., \( L \setminus (L \Join_{\text{second}} L) \), with second\((s, s') = s' \). In what follows, this expression is called bestSessions\((\succeq, L)\).

Note that intersection can be simulated with the join operation. Indeed, \( L \setminus L' = L \Join_{\text{same, first}} L' \setminus \text{for any two sessions } s, s', \text{first}(s, s') = s \text{ and same}(s, s') \text{ if } s = s' \).

Note also that this join operation is not symmetric unless \( \theta \) is symmetric and \( f(s, s') = f(s', s) \) for all \( s, s' \). Moreover, the \( f \) function is needed for closeness reason. In that sense, it is inspired by the join operation defined in [2] for joining cubes.

5.2.4 Grouping and aggregation

This operator allows to group sessions that are related to one another through relation \( \theta \), and to aggregate them using an aggregation function \( \text{agg} \).

**Definition 5.4.** Let \( L \) be a log, \( \text{agg} \) be a function aggregating a set of sessions into a session, and \( \theta \) be a binary relation over sessions.

\[
\pi_{\theta, \text{agg}}(L) = \{ \text{agg}\{s' \in L | \theta(s, s')\} | s \in L \}.
\]

**Example 5.4.** This operation can be used to transform sessions. For instance, extracting from a log \( L \) the last query of each session can be expressed by: \( \pi_{\text{same, last}}(L) \) where, for any two sessions \( s, s' \), it is same\((s, s') \) if \( s = s' \), and last\(((q_1, \ldots, q_n)) = \{q_n\} \). In what follows, this expression is called extractLast(L). If \( \text{removeLast}((q_1, \ldots, q_n)) = (q_1, \ldots, q_{n-1}) \), is used instead of last, then the expression modifies each session of \( L \) by removing the last query. This expression will be called extractAllButLast(L) in what follows.

This operation can also be used for extracting the best queries from a log \( L \) in the sense of a partial order relation \( \succeq \) over queries:

\[
\pi_{\theta, \text{best, getBest}}(L) \text{ where, } \text{better}(s, s') \text{ holds only if } s \text{ contains at least one of the best queries of } L' \text{ and } s' \text{ does not, and getBest}(S) = \{q_i \in S \text{ such that the better(s, s') holds for all } s, s' \in L \} \text{ and } \text{agg}(S) = \{q_i \} \text{ where for all } i, j \in \{1, \ldots, n\}, \text{ it is } i < j \text{ if either } q_i \text{ and } q_j \text{ belong to the same session of } S \text{ and } s^{-1}(q_i) < s^{-1}(q_j) \text{ or, they belong to different sessions } s \text{ and } s' \text{ respectively, and prefix}(s, s') \text{ holds. This expression is called bestQuery}(\succeq, L) \text{ in what follows.}
\]

5.3 Properties

We briefly review the properties of the language in terms of closure, completeness and minimality.

It can easily be seen that the language is closed under composition, the result of any operation being a set of sessions.

The language is complete. Indeed, for any pair of logs \( L \) and \( L' \), there is an expression that enables to transform \( L \) into \( L' \). The transformation would proceed as follows:

1. pick a session \( s \) in \( L \) with \( \sigma \),
2. transform it into a session \( s' \) of \( L' \) using \( \pi \),
3. repeat the previous steps until all sessions of \( L' \) are formed,
4. union all transformed sessions to form \( L' \).

The language is not minimal since \( \sigma \) can be simulated with \( \Join \). Indeed, \( \sigma_{\theta, \Join}(L) = L' \Join_{\theta, f, \Join} L \) where \( L' = \{s\} \) and \( f(s, s') = s' \). It can easily be seen that removing \( \sigma \) makes the language minimal.

6. ADVANCED MANIPULATIONS

Finally, we illustrate the language with some expressions describing various user-centric tasks.

6.1 Summarizing and generalizing a log

The group-by and aggregate operation can be used to summarize a log. For instance, given the equivalence relation \( \text{sim} \) defined in Section 4.2, sessions can be grouped together using \( \text{sim} \), and then aggregated using the session in each group that minimizes the sum of the distances to other sessions of the group, with expression: \( \pi_{\text{lim}, \text{rep}}(L) \), where \( \text{rep}(S) = \arg \min_{s \in S} \{\sum_{s' \in S} \text{dist}(s, s')\} \), \( \text{dist} \) being the distance used in the definition of \( \text{sim} \).

A log can be generalized by computing the most specific ancestors to the sessions it contains. The expression \( E_1 = L \Join_{\text{all, same}} L \), where all\((s, s') \) holds for all \( s, s' \in L \), gives the most specific ancestors of all pairs of sessions in \( L \). Then \( E_2 = E_1 \setminus \text{find}(\text{E}^*_{1}) \) removes the sessions in \( S' \). Finally, \( \text{bestSessions}(E_2, \succeq) \), gives the most general sessions of \( E_2 \). Calling this expression \( \text{generalize}(L) \), more general logs can be expressed by: \( \text{generalize}(...(\text{generalize}(L)...) \).

6.2 Personalization

Obviously, the language can be used to express dominance queries. For instance, assuming a preference relation \( \text{pref} \) over sessions, extracting the preferred sessions from a log \( L \) is: \( \text{bestSessions}(\text{pref}, L) \). Besides, assuming a preference relation \( \text{pref}_q \) over queries, extracting the preferred queries from a log \( L \) is: \( \text{bestQueries}(\text{pref}_q, L) \).

Obtaining a total order over queries from a total order over sessions can be expressed in the language. Assuming a total order \( \text{pref} \) over queries, the queries of a log can be ordered in a singleton where the order of queries in the session reflects the order over sessions. More precisely, this is achieved with \( \pi_{\theta, \text{agg}}(L) \) with \( \theta(s, s') \) for all \( s, s' \in L \), and \( \text{agg}(S) = \{q_1, \ldots, q_n\} \) where for all \( i, j \in \{1, \ldots, n\} \), it is \( i < j \) if either \( q_i \) and \( q_j \) belong to the same session of \( S \) and \( s^{-1}(q_i) < s^{-1}(q_j) \) or, they belong to different sessions \( s \) and \( s' \) respectively, and \( \text{pref}(s, s') \) holds. This expression is called order\((\text{agg}, L)\).

Finally, note that a partial order over queries can be expressed for the queries of a log \( L \) as follows. First, separate all queries of the log by breaking all the sessions: \( L' = \text{extractLast}(L) \cup \text{extractLast}(...(\text{extractAllButLast}(L)...) \cup \ldots \). Then, use the join operation to form pairs of queries (i.e., sessions of length 2) to represent the order over the queries.
of $L'$: $L' \triangleright_{\theta, f} L'$ with $\theta((q), (q'))$ if a given relation between $q$ and $q'$ holds (like for instance $q \supseteq q'$ or $q \subseteq q'$) and $f((q), (q')) = (q, q')$.

6.3 Query recommendation

The language can be used to describe various query recommendation approaches. For instance, in the approach described in [6], the last queries of past sessions that are similar to a given session, are recommended. This approach can be expressed, for a log $L$, by: $\text{extractLast}(\text{findSimilar}(\theta, s, L))$ where $\theta$ is based on an extension of the edit distance to compare sessions. Moreover, these recommendations can be ordered using the expression $\text{order}(\text{agg}, L)$ defined above, where $\text{agg}$ uses a total order over sessions derived from a total order over queries.

7. CONCLUSION

This paper presented a language for manipulating logs of OLAP queries, defined as sets of sessions. The language relies on binary relations over sessions, and we give some examples of such relations. We illustrate the use of the language for describing various user-centric approaches, like query recommendation or log summarization, that are particularly relevant in the context of data warehouse exploration by means of OLAP sessions.

This language is a preliminary contribution to the foundations of query log management. Future works include using the framework to investigate logical properties of log manipulations, as well as its use in other context, for instance to manipulate web logs.

8. REFERENCES


