Wolfgang Lehner, Wolfgang Hummer, Lutz Schlesinger

Processing Reporting Function Views in a Data Warehouse Environment

Erstveröffentlichung in / First published in:

Proceedings 18th International Conference on Data Engineering. San Jose, 26.02-01.03.2002. IEEE, S. 176-185. ISBN 0-7695-1531-2

DOI: https://doi.org/10.1109/ICDE.2002.994707

Final edited form was published in "Proceedings 18th International Conference on Data Engineering", San Jose, 2002. ISBN 0-7695-1531-2.
https://doi.org/10.1109/ICDE.2002.994707

Processing Reporting Function Views in a Data Warehouse Environment

Wolfgang Lehner, Wolfgang Hümmer, Lutz Schlesinger

University of Erlangen-Nuremberg (Database Systems)
Martensstr. 3, Erlangen, 91058, Germany
EMail: {lehner, huemmer, schlesinger}@informatik.uni-erlangen.de

Abstract

Reporting functions reflect a novel technique to formulate sequence-oriented queries in SQL. They extend the classical way of grouping and applying aggregation functions by additionally providing a column-based ordering, partitioning, and windowing mechanism. The application area of reporting functions ranges from simple ranking queries (TOP(n)-analyses) over cumulative (Year-To-Date-analyses) to sliding window queries. In this paper we discuss the problem of deriving reporting function queries from materialized reporting function views, which is one of the most important issues in efficiently processing queries in a data warehouse environment. Two different derivation algorithms including their relational mappings are introduced and compared in a test scenario.

1. Introduction

A data warehouse together with OLAP and data mining tools reflects an asset for every organization in the decision making process. While recent work in optimizing database technology for this application area has concentrated on specifying complex grouping conditions (like `CUBE()`, `ROLLUP()`, and `GROUPING SETS()`; [5]) and materialized view design (e.g. [9], [18], [3],...), more complex statistical analyses often require sequence semantics. Such applications range from simple ranking queries and Year-To-Date analyses to complex pattern matching and similarity algorithms. In summary, sequence processing is demanded by many applications, and is – from a complexity perspective – far beyond the simple computational model of OLAP.

Overview of Reporting Functions

Recent developments consider these requirements by introducing the SQL extension of *Reporting* (or *OLAP*) *Functions* ([10], [13], [17]) extending aggregation by column-wise partitioning, ordering, and optional dynamic windowing. The following example illustrates the concept of reporting functions with regard to the tables `c_transactions` holding credit card transactions and `l_locations` with a mapping of shops to geographic information like cities and regions. The query below returns the actual date, the amount of each transaction and four reporting function columns for customer `4711`.

```
SELECT c_date, c_transaction,
   SUM(c_transaction) OVER     -- overall cumulative sum
  ( ORDER BY c_date ROWS UNBOUNDED PRECEDING ) AS cum_sum_total,
   SUM(c_transaction) OVER     -- cumulative sum per month
  ( PARTITION BY month(c_date) ORDER BY c_date
    ROWS UNBOUNDED PRECEDING ) AS cum_sum_month,
   AVG(c_transaction) OVER     -- centered 3 day moving average
  ( PARTITION BY month(c_date), l_region ORDER BY c_date
    ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) AS c_3mvg_avg,
   AVG(c_transaction) OVER     -- prospective 7 day moving average
  ( ORDER BY c_date
    ROWS BETWEEN CURRENT ROW AND 6 FOLLOWING) AS c_7mvg_avg
 FROM     c_transactions, l_locations
WHERE     c_locid = l_locid AND c_custid = 4711;
```

Reporting Functions are defined with the `OVER()`-clause following a regular aggregation function thus algorithmically declaring the scope of the aggregation function for each single incoming tuple. According to [10] and [13], figure 1 gives the corresponding general syntax diagram. The `ORDER BY`-clause of a reporting function determines the local sort order of each column independent of any optional global `ORDER BY`-clause. The scope of the aggregation function is determined by a window aggregation group specification. For example, the first two reporting function columns of the example above exhibit a scope from the 'first' to the current tuple, thus implementing cumulative semantics. The notion of a 'first' tuple may optionally be further refined by applying a partition clause: The pointer to the 'first' tuple is reset for each new partition defined in the `PARTITION BY`-clause, so that the running sum of the credit card transactions are computed on a monthly basis in the second reporting function. The columns `c_3mvg_avg` and `c_7mvg_avg` demonstrate the specification of sliding windows: To compute a single output value for `c_3mvg_avg`, the left, the current, and the right tuples are considered to retrieve the local average value. Smoothing of sequence data is one of the most relevant applications for sliding windows. It is important to notice here that in opposite to regular aggregation functions in the context of a global group-by, reporting functions do not shrink the data volume but produce one output value for each single input value.

Related Work

Sequence processing is one of the most intensively studied problems in database technology. Different proposals range from special data models like SEQ ([16]) over the introduction of special sequence operators ([19], [7]) to

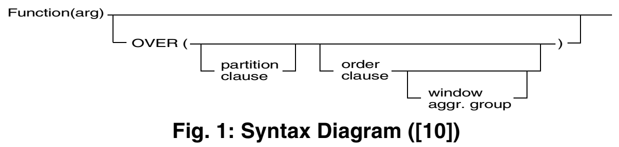

Fig. 1: Syntax Diagram ([10])

complex algorithms like similarity search or pattern recognition operating in a special sequence-oriented environment. However, tightly integrating sequence processing techniques into the relational context and considering the derivation problems in the presence of materialized views has not been addressed in prior work.

The overall processing strategy in evaluating reporting functions is the following: In a first step, (after performing necessary joins and selections) an optionally existing global group by-operation is executed. Specifying queries with complex grouping conditions ([5]) and deriving queries from views at that level of detail reflect a very well studied problem (e.g. [20]) and are not further discussed in the context of the paper. The second and third step in evaluating reporting functions consists in defining column-wise partitions and sorting the data within each partition. Again, considering sort orders in the context of derivation problems is a well understood problem and discussed for example in [6], [1], or [14]. However, deriving queries from materialized data with an additional window specification within each partition is still an open issue, which at the same time is discussed in this paper.

Contribution and Organization of the Paper

Reporting functions reflect an important piece in an analytical environment, needed to exploit the benefit of integrated data usually gathered in a data warehouse. However, implementing sequence operations within a relational engine causes tremendous effort, because traditional operator design is not well suited for caching data to compute cumulative sequences or grouping data on a column basis. Hence this paper presents a comprehensive study of dealing with sequence data from two perspectives. On the one hand, given recursive algorithms may be used in the presence of special techniques like internal caches. On the other hand, we always give an operator pattern of each algorithm, specified within the pure relational model. Although not very efficient, this approach provides at least an easy way of implementing reporting functionality useful for cutting down developing time and costs. The given operator patterns may be applied in query rewrite directly after parsing the query exhibiting a reporting function.

The following section formally introduces the concept of sequences, outlines strategies to compute sequence values, and gives rules to incrementally maintain materialized sequence views. Section 3 presents basic techniques for working with cumulative and sliding window sequences. Section 4 and section 5 address the problem of deriving sliding window queries from materialized views defined over sequences with a different window size. After that, section 6 gives a short insight in generalizing the previous concepts for reporting function sequences involving ordering and partitioning. Section 7 finally presents some empirical evaluations of the algorithms explained in section 4 and section 5. The paper closes with a summary and conclusion in section 8.

2. Simple Reporting Function Sequences

As shown in the previous example, a single reporting function represents a sequence specification. This section classifies different sequence types and formalizes the operations.

2.1. Definition of Sequences

Without loss of generality we assume that raw data values x_i only exist for positive values of i. For other i x_i is set to zero.

Definition: *Simple Sequence*

A simple sequence $\tilde{x}$ is defined as a triple (S, W, F_A) where S is a tuple (S_L, S_H) specifying start and stop positions of the sequence. W is a tuple (W_L, W_H) defining the operational scope of each position; W_L and W_H are a list of functions computing the lower and upper bound of the window at each sequence position, i.e. $W_L = <w_L(1), w_L(2), ..., w_L(n)>$ and $W_H = <w_H(1), w_H(2), ..., w_H(n)>$. F_A is a regular aggregation function computing the result of each set of input values of a given window.

The previous definition implies that the result of the reporting function defined by a simple sequence (S, W, F_A) at position k ($S_L \leq k \leq S_H$) is computed by $\tilde{x}(k) = \tilde{x}_k = F_A\{x(w_L(k)), x(w_L(k)+1), ..., x(w_H(k)-1), x(w_H(k))\}$. Without loss of generality, we furthermore assume that a simple sequence starts at position $S_L=1$ and ends at position $S_H=n$, where n is the cardinality of the sequence. The window size of a sequence at a position k is then defined by $W(k) = 1+w_H(k)-w_L(k)$.

In general, we may distinguish two different types of simple sequences with regard to the shape of the specified window:

- *Cumulative Windows:* The range of cumulative windows (for example to compute year-to-date queries) is algorithmically determined by $w_L(k)=0$ and $w_H(k)=k$ ($1 \leq k \leq n$), so that the window size at position k is $W(k)=1+W(k-1)$ for $k>1$ and $W(1)=1$.
- *Sliding Windows:* Sliding windows are defined by a fixed relative distance to the lower and upper bound of the window at a specific position, so that $w_L(k)=k-l$ and $w_H(k)=k+h$ with l and h as arbitrary but constant integer values. Obviously the window size yields $W(k)=1+l+h$ ($1 \leq k \leq n$).*

* For easier handling, we assume that $l \geq 0$, $h \geq 0$, and $l+h>0$. This implies that the lower bound is left of k and the upper bound is located right of k.

Since the handling of sliding windows is more complex than cumulative windowing and all techniques proposed for sliding windows are easily transferable to cumulative windowing sequences, we concentrate on sliding window sequences defined by *(l, h)* with $l \geq 0, h \geq 0$ starting at position *1* with *n* input values x_i $(1 \leq i \leq n)$ in the following discussions. From the set of aggregation functions (SUM, COUNT, AVG, MIN, MAX), we emphasize the SUM aggregation function, because COUNT is trivial (either constant or the current position) and AVG may be directly derived from SUM and COUNT. The semi-algebraic functions MIN and MAX are mentioned whenever the application is permitted.

This means that the value of the sequence $\tilde{x}$ at a certain point *k* is defined as the sum of all values x_i from the lower bound (x_{k-l}) up to the upper bound of the window (x_{k+h}). Because of our earlier assumption it is clear that $\tilde{x}_k = 0$ for $k \leq -h$ and $k > n+l$

2.2. Computing Sequences

The naive way of computing sequence data is to follow the explicit form, i.e. $\tilde{x}_k = F_A\{x(w_L(k)), x(w_L(k)+1), ..., x(w_H(k)-1), x(w_H(k))\}$. A relational mapping without explicit support of reporting functionality inside the relational engine is depicted in figure 2 for the sample query computing a centered sliding window of size 3.

```
SELECT pos, SUM(val) OVER (ORDER BY pos
              ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)
  FROM seq;
```

As can be seen, simulating sequence-processing within the relational model requires the execution of a self join of the underlying table. Computing sequences can be sped up as soon as special operators are available. In case of a cumulative sequence, it is obvious that the sequence value $\tilde{x}_k$ can be computed by referring to the preceding value and the raw data at the current position, i.e.

$$\tilde{x}_k := \tilde{x}_{k-1} + x_k$$

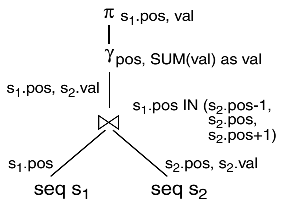

Fig. 2: Relational Mapping

For computing sliding window sequences, we take into account that two neighbored windows at position *k* and *k-1* exhibit the following algebraic relationship (figure 3):

$$\tilde{x}_k + x_{k-l-1} = \tilde{x}_{k-1} + x_{k+h}$$

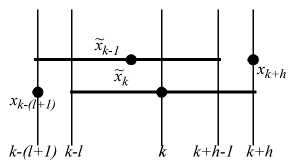

Fig. 3: Sequence Relationship

Besides other applications, which are exploited in the following sections, we may use this relationship to compute sequence data in a pipe-lined style using the following recursion:

$$\tilde{x}_k := \tilde{x}_{k-1} + x_{k+h} - x_{k-l-1}$$

Instead of performing *w(k)+1* operations at every position to compute the sequence value, referring to the given relationship requires only three operations independent of the window size. The cache needs the size of *w(k)+2* holding $\tilde{x}_{k-1}$ and x_{k-l-1} up to x_{k+h}.

2.3. Maintaining Materialized Sequence Data

One of the most characteristic properties of a data warehouse environment from a database perspective is that materialized views are stored redundantly in the data warehouse database to speed up incoming queries. In order to synchronize materialized views with changes made to the base tables, views require either a full recomputation or an incremental update. Updating materialized views in general is a well-studied problem ([8], [11]). However, incrementally maintaining materialized sequence data in the context of reporting functions was not considered in prior work. The following list provides a set of rules to maintain materialized sliding window sequence data:

- *Update Operation:* The update operation changes the value $\tilde{x}_k$ at a single position *k* to $\tilde{x}_k'$. All sequence values which are in the scope of the modified raw data value have to be updated, so that the general formula for the new sequence values $\tilde{x}_i'$ $(1 \leq i \leq n)$ of $\tilde{x}'$ results in[†]:

$$\tilde{x}_i' = \begin{cases} \tilde{x}_i & 1 \leq i < k-h \\ \tilde{x}_i - x_k + x_k' & k-h \leq i \leq k+l \\ \tilde{x}_i & i > k+l \end{cases}$$

- *Insert Operation:* Inserting a new raw data value implies that an additional value is inserted at position *k* so that the positions of all values right of *k* are incremented. With *w=l+h+1*, the sequence values change according to the following rule:

$$\tilde{x}_i' = \begin{cases} \tilde{x}_i & 1 \leq i < k-h \\ \tilde{x}_i + x_k - x_{i+w-1} & k-h \leq i \leq k \\ \tilde{x}_{i-1} + x_k - x_{i-w+1} & k \leq i \leq k+l \\ \tilde{x}_{i-1} & i > k+l \end{cases}$$

- *Delete Operation:* For deleting a value x_k from a sequence, the necessary modifications are reflected by the following rule:

$$\tilde{x}_i' = \begin{cases} \tilde{x}_i & 1 \leq i < k-h \\ \tilde{x}_i - x_k + x_{i+w-2} & k-h \leq i < k \\ \tilde{x}_{i+1} - x_k + x_{i+w-1} & k \leq i < k+l \\ \tilde{x}_{i+1} & i \geq k+l \end{cases}$$

† Obviously materialized sequence views defined over MIN/MAX aggregation functions are incrementally updateable with $\min(\tilde{x}_i, x_k')$ resp. $\max(\tilde{x}_i, x_k')$ for all affected positions.

As can be seen, incrementally updating sequence data is more efficient than recomputing the whole sequence, because only the affected values have to be recomputed. Even in the presence of insert and delete operations, the changes to the sequence data remain local.

3. Derivability of Sequence Data

Storing materialized views with reporting functions requires that incoming queries are able to take advantage of the existence of the materialized views and can be rewritten by utilizing these views. This section introduces the concept of derivability of sequence queries from materialized sequence data. Although, there is no evident advantage with regard to the cardinality of materializing sequence data in opposite to the classical group-by problem ([12]), we identify many applications making the derivation of sequence queries necessary. For example, a data warehouse system may propose a caching strategy of incoming user queries ([4], [21], [15]) to avoid the costly process of explicitly computing candidates of materialized views ([2], [9]). If users are heavily relying on sequence processing and the system is not able to consider the derivation of sequence queries from materialized sequence views, no support can be achieved.

A first solution of the problem of deriving sequence queries from materialized sequence data would be to reconstruct the original raw values using the recursive formula given below. Although this might be a feasible approach if the engine supports internal caching and special sequence operators, we additionally propose explicit forms of deriving sequence queries, which may be easily incorporated into relational database engines without much internal rewriting.

3.1. Materialized Cumulative Sequences

This section addresses the problem of deriving single data points of sequence values if the underlying sequence is of type cumulative.

Reconstruction of Raw Data Values

A single raw data point x_k may be derived from a cumulative sequence by referring to the relationship of two directly neighbored sequence values, i.e.

$$x_k := \tilde{x}_k - \tilde{x}_{k-1}$$

A pure relational mapping again requires a self join of the materialized sequence table to retrieve the original data values (figure 4). A `CASE()`-statement is used to negate the sequence value at position *k-1*, so that a SUM aggregation function with grouping over the sequence positions yields the result.

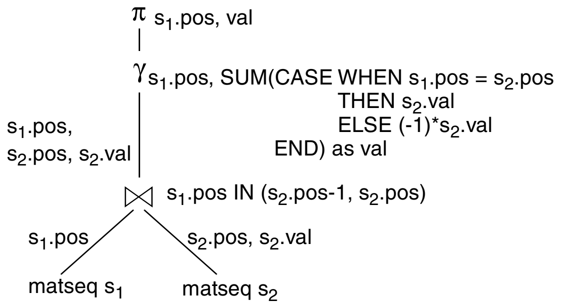
π s1.pos, val
γ s1.pos, SUM(CASE WHEN s1.pos = s2.pos THEN s2.val ELSE (-1)*s2.val END) as val
s1.pos, s2.pos, s2.val
⋈ s1.pos IN (s2.pos-1, s2.pos)
s1.pos
s2.pos, s2.val
matseq s1
matseq s2

Fig. 4: Reconstructing Raw Data Values

Derivation of Sliding Window Sequences

To derive a sliding window sequence $\tilde{y}_k$ defined by *(l, h)* from $\tilde{x}$ needs two integers *n* and *m* so that $\tilde{x}_m$ is right justified with $\tilde{y}_k$ and that $\tilde{x}_n$ is summing up everything to the left of $\tilde{y}_k$. This idea is clarified in figure 5 and yields the following result:

$$\tilde{y}_k := \tilde{x}_{k+h} - \tilde{x}_{k-l-1}$$

This formula holds even for small values of *k*, because $\tilde{x}_{k-l-1}$ is defined to be *0* for $k<l+1$. In this case, only the first part of the difference contributes to the final result. The relational mapping corresponds to figure 4 with adaptations needed in the sequence positions to compute the new sequence value $\tilde{y}_k$.

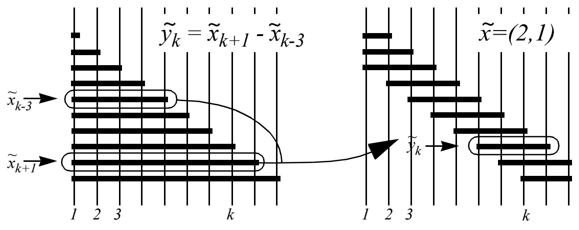

Fig. 5: Sliding Windows from a Cumulative Window Sequence

3.2. Materialized Sliding Window Sequences

In case of deriving data from materialized sliding window sequences, we may refer to the relationship of two directly neighbored sequence values. This relationship may be used to reconstruct the original data values, so that x_k at position *k* is computed using the following expression:

$$x_k := \tilde{x}_{k+l} - \tilde{x}_{k+l+1} + x_{k+l+h+1}$$

resp. $x_k := \tilde{x}_{k-h} - \tilde{x}_{k-h-1} + x_{k-l-h-1}$

As we will see in the following example, the only necessary prerequisite for this mechanism to work correctly is the requirement of completeness, i.e. the existence of the header and trailer of sequence $\tilde{x}$. The given recursion can be transformed (see section 5) into the following explicit form with $w=W(k)=l+h+1$ denoting the window size of the materialized sliding window sequence $\tilde{x}$:

$$x_k := \underset{i \geq 0}{\_\_}(\tilde{x}_{k-h-iw} - \tilde{x}_{k-h-1-iw})$$

The summation can be stopped after i_{up}, because beyond this point the larger value $\tilde{x}_{k-h-iw}$ yields zero. The parameter i_{up} is the minimal i fulfilling the following condition:

$\tilde{x}_{k-h-i_{up}w} = 0 \Leftarrow k-h-i_{up}w \leq -h$ and therefore $i_{up} \geq \frac{k}{w}$.

For example i_{up} can be set to: $i_{up} = \left\lceil \frac{k}{w} \right\rceil$

Derivation of Sliding Window Sequences

To illustrate the problem of deriving sliding window queries from a materialized sliding window view, consider the following example: Given a materialized sequence $\tilde{x}=(2,1)$ and a sequence query $\tilde{y}=(3,1)$ we may obviously deduce from figure 6 that the values correspond to each other at the first three positions. In addition to $\tilde{x}_4$, $\tilde{y}_4$ also comprises x_1, the first raw data value, i.e. $\tilde{y}_4 = \tilde{x}_4 + x_1$. To avoid raw data access for x_1, we have to use the corresponding $\tilde{x}$ sequence value, in this case $\tilde{x}_0$. Analogous, $\tilde{y}_5$ can be computed from $\tilde{x}_5$ and x_2. Since x_2 is unfortunately not accessible, we may again substitute x_2 by $(\tilde{x}_1 - x_1)$ which is the same as $(\tilde{x}_1 - \tilde{x}_0)$. Using the same strategy we are able to retrieve $\tilde{y}_6$, $\tilde{y}_7$, and $\tilde{y}_8$ by $(\tilde{x}_2 - \tilde{x}_1)$, $(\tilde{x}_3 - \tilde{x}_2)$, and $(\tilde{x}_4 - \tilde{x}_3)$. Things are getting more complicated if we look at position 9 and higher, where an additional compensation term is necessary.

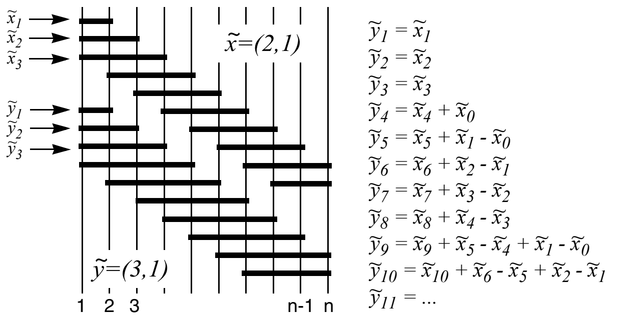
$\tilde{x}_1$
$\tilde{x}_2$
$\tilde{x}_3$
$\tilde{y}_1$
$\tilde{y}_2$
$\tilde{y}_3$
$\tilde{x}=(2,1)$
$\tilde{y}=(3,1)$
1 2 3
n-1 n
$\tilde{y}_1 = \tilde{x}_1$
$\tilde{y}_2 = \tilde{x}_2$
$\tilde{y}_3 = \tilde{x}_3$
$\tilde{y}_4 = \tilde{x}_4 + \tilde{x}_0$
$\tilde{y}_5 = \tilde{x}_5 + \tilde{x}_1 - \tilde{x}_0$
$\tilde{y}_6 = \tilde{x}_6 + \tilde{x}_2 - \tilde{x}_1$
$\tilde{y}_7 = \tilde{x}_7 + \tilde{x}_3 - \tilde{x}_2$
$\tilde{y}_8 = \tilde{x}_8 + \tilde{x}_4 - \tilde{x}_3$
$\tilde{y}_9 = \tilde{x}_9 + \tilde{x}_5 - \tilde{x}_4 + \tilde{x}_1 - \tilde{x}_0$
$\tilde{y}_{10} = \tilde{x}_{10} + \tilde{x}_6 - \tilde{x}_5 + \tilde{x}_2 - \tilde{x}_1$
$\tilde{y}_{11} = \ldots$

Fig. 6: Sample derivation process

Interesting to see is that we are generally able to retrieve sequence queries from materialized sequence data, if – in the special case above – we have access to the first atomic value. Moreover, the computation rules show a regular pattern which might be exploited to speed up the computation. The necessary prerequisites in general, the formal specification of the derivation rules and an algebraic proof are subject to the following discussions of section 4 and section 5.

Header and Trailer of a Sequence

One of the lessons learnt from the above example was the need of 'some' data outside the sequence specification, which leads to the notion of a complete sequence.

Definition: *Complete Simple Sequence (CSS)*
A simple sequence is complete if the sequence representation exhibits a sequence header and a sequence trailer. A header consists of all sequence values for positions ranging from $-\infty$ to 0; analogous, a trailer of a sequence is defined as the set of sequence values for positions ranging from $n+1$ to ∞.

As shown in figure 7, the interesting part of a header/trailer consists of the sequence values for positions $-h+1$ to 0 ($\tilde{x}_0$ in the example) and $n+1$ to $n+l$ ($\tilde{x}_{n+1}$ and $\tilde{x}_{n+2}$ in the example), because given raw data values at the positions of 1 to n still contribute to these sequence values. Unspecified raw data values at header and trailer positions are set to $x_k=0$ ($k<1$, $k>n$). A sequence is said to be left-bounded, if no preceding value contributes to the window, i.e. $l=0$, and right-bounded, if $h=0$.

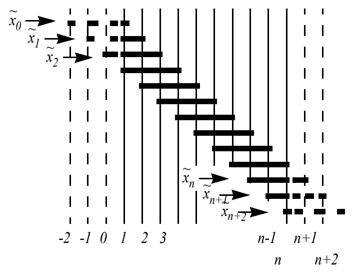

Fig. 7: Sample Complete Sequence

4. The MaxO Algorithm

Deriving sliding window queries from materialized sequence data may be achieved in many ways. We propose two alternative algorithms which are subject of description in this and the following section. The MaxOA, which stands for "*Maximal Overlapping Algorithm*" tries to retrieve a new sequence value $\tilde{y}_k$ with a coarser window granularity by referring to materialized sequence values $\tilde{x}_k$ providing maximal overlapping.

Preconditions

Although for merely technical reasons, we assume that the window size of the query must not be larger than twice the window size of the materialized sliding window sequence:

(I) $w'_L(k) >= w_L(k')$

(II) $w_H(k') = 1 + w_L(k)$,

where $w()$ denotes the window positions of sequence $\tilde{x}=(l_x,h_x)$ and $w'()$ represents the window specification of sequence $\tilde{y}=(l_y,h_y)$ reflecting the user query. Given these equations, we may deduce that – with the same upper bound h – the lower bound of $\tilde{y}$ is restricted to $l_y \leq h-1+2l_x$.

4.1. Derivation Pattern for Sequences Sharing a Common Bound

The first step in tackling the problem of deriving sequence queries from materialized sequence data focusses on sequences sharing either the lower or upper bound, i.e. the l or h parameter within the window definition. Without loss of generality, we assume that both sequences exhibit the same parameter $h = h_x = h_y$, so that the materialized sequence $\tilde{x}$ is defined as $\tilde{x}=(l_x,h)$ and the deriving sequence may be specified as $\tilde{y}=(l_y,h)$.

The basic procedure to retrieve a value of $\tilde{y}_k$ from sequence values of $\tilde{x}$ is to add $\tilde{x}_k$ and $\tilde{x}_{k'}$ with $k' = k-(l_y-l_x)$, so that the lower bound of $\tilde{x}_{k'}$, $w_L(k')$, corresponds to the lower bound of $\tilde{y}_k$, i.e. $w'(k)=l_y$.

Definition: *Coverage Factor* Δl

The coverage factor $\Delta l = l_y-l_x > 0$ denotes the shift distance which is required to perform a coverage for values of the derived sequence.

Adding sequence values $\tilde{x}_k$ and $\tilde{x}_{k-\Delta l}$ introduces an error term as soon as the window size of $\tilde{y}$ is less than $2W_x$. The error term, denoted $\tilde{z}_k$ in figure 8, is again a regular sequence of type $(l_x,h-\Delta l)$, called compensation sequence. Computing the new value for $\tilde{y}_k$ is reduced to the evaluation of the corresponding compensation sequence, i.e.

$$\tilde{y}_k := \tilde{x}_k + \tilde{x}_{k-\Delta l} - \tilde{z}_k.$$

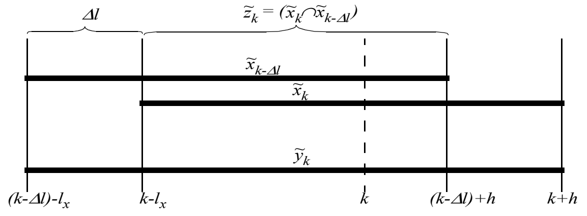

Fig. 8: Coverage using a Compensation Sequence

Definition: *Compensation Sequence* $\tilde{z}$

A compensation sequence $\tilde{z}$ represents the error term $\tilde{z}_k$ created by deriving a value $\tilde{y}_k$ by adding $\tilde{x}_k$ and $\tilde{x}_{k-\Delta l}$.

To compute the error term $\tilde{z}_k$ of a compensation sequence, we rely on the same principle as for computing a sequence in a pipelined fashion (section 2.2). Figure 9 illustrates the relationship of $\tilde{z}_k$ and $\tilde{x}_{k-\Delta l}$ of the following equation:

$$\tilde{z}_k + \tilde{x}_{k-(\Delta l+\Delta p)} = \tilde{z}_{k-(\Delta p+\Delta l)} + \tilde{x}_{k-\Delta l}$$

The parameter Δp denotes the number of positions needed, so that two values from the materialized sequence $\tilde{x}$, namely $\tilde{x}_{k-\Delta l}$ and $\tilde{x}_{k-(\Delta l+\Delta p)}$, overlap in exactly $\Delta l-1$ positions.

Definition: *Overlap Factor* Δp

The overlap factor Δp of a compensation sequence determines the number of positions used to establish a direct relationship between entries of the compensation sequence and the original sequence data.

The overlap factor may be retrieved from the distance of the upper bound of the sequence window of $\tilde{x}_{k-(\Delta l+\Delta w)}$ and the lower bound of the sequence value $\tilde{x}_{k-\Delta l}$ (Figure 9):

$$w_H(k-(\Delta l+\Delta p)) - w_L(k-\Delta l) = \Delta l - 1$$
$$k - (\Delta l + \Delta p) + h - ((k - \Delta l) - l_x) = \Delta l - 1$$

and therefore

$$\Delta p = 1+l_x+h-\Delta l$$

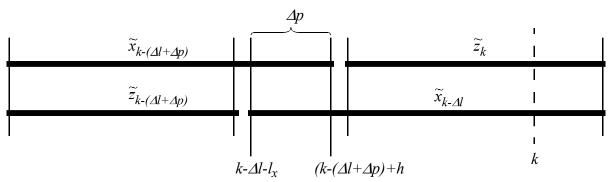
Δp
$\tilde{x}_{k-(\Delta l+\Delta p)}$
$\tilde{z}_k$
$\tilde{z}_{k-(\Delta l+\Delta p)}$
$\tilde{x}_{k-\Delta l}$
$k-\Delta l-l_x$
$(k-(\Delta l+\Delta p)+h$
k

Fig. 9: Overlap Factor

Summary of Sequence Derivation

Having constructively generated the single steps of deriving a sequence query from materialized sequence data, we may now give a general rule for deriving sequence values from materialized sliding windows.

Theorem: *Sequence Derivation (MaxOA)*

A sequence $\tilde{y}=(l_y,h)$ is derivable from a complete sequence $\tilde{x}=(l_x, h)$ using the following algorithm ($l_y \leq h-1+2l_x$):

$$\tilde{y}_k := \tilde{x}_k + \tilde{x}_{k-\Delta l} - \tilde{z}_k,$$

where $\tilde{z}_k$ is the compensation sequence defined as $\tilde{z}=(k-w_L(k), w_H(k-\Delta l)-k)$ and computed by the following rule:

$$\tilde{z}_k := \tilde{x}_{k-\Delta l} - \tilde{x}_{k-(\Delta l+\Delta p)} + \tilde{z}_{k-(\Delta l+\Delta p)}$$

The coverage factor Δl and the overlap factor Δp are computed by:

$$\Delta l = l_y-l_x > 0$$
$$\Delta p = 1+l_x+h-\Delta l$$

Clarification: The proof of the derivation theorem is intentionally omitted due to space restrictions. However, we refer to the constructive way of introducing the formula.

Explicit From versus Recursive Form

Since we are focussing on implementing sequence derivation within a relational environment, it might be advisable to find a proper explicit form of the derivation formula given above.

Theorem: *Explicit Form of Sequence Derivation*

A sequence $\tilde{y}=(l_y,h)$ is derivable from a complete sequence $\tilde{x}=(l_x, h)$ according to the following explicit formula ($l_y \leq h-1+2l_x$):

$$\tilde{y}_k := \tilde{x}_k + \sum_{i=1}^{\infty} \tilde{x}_{k-i(\Delta l+\Delta p)} - \sum_{i=1}^{\infty} \tilde{x}_{k-((i+1)\Delta l+i\Delta p)}$$

Proof: We proof the theorem by induction:

(a) $k=1$: obvious because $\tilde{y}_1 = \tilde{x}_1$

(b) Due to the window-based character, we are not able to prove the theorem directly from k to $k+1$. Instead we have to prove this transition from k to $k+(\Delta l+\Delta p)$, from $k+1$ to $k+(\Delta l+\Delta p)+1$, up to $k+(\Delta l+\Delta p)-1$ to $k+2(\Delta l+\Delta p)$.

- k to $k+(\Delta l+\Delta p)$:

$$\tilde{y}_{k+(\Delta l+\Delta p)} = \tilde{x}_{k+(\Delta l+\Delta p)} +$$
$$\sum_{i=1}^{\infty} \tilde{x}_{k-(i-1)(\Delta l+\Delta p)} - \sum_{i=1}^{\infty} \tilde{x}_{k-(i\Delta l+(i-1)\Delta p)}$$

$$= \tilde{x}_{k+(\Delta l+\Delta p)} + \tilde{x}_k + \sum_{i=1}^{\infty} \tilde{x}_{k-i(\Delta l+\Delta p)} - \tilde{x}_{k-\Delta l} - \sum_{i=1}^{\infty} \tilde{x}_{k-((i+1)\Delta l+i\Delta p)}$$

$$= \tilde{y}_k + \tilde{x}_{k+(\Delta l+\Delta p)} - \tilde{x}_{k-\Delta l}$$

$$(*) = (\tilde{x}_k + \tilde{x}_{k-\Delta l} - z_k) + \tilde{x}_{k+(\Delta l+\Delta p)} - \tilde{x}_{k-\Delta l}\ (+\tilde{x}_{k-\Delta p} - \tilde{x}_{k-\Delta p})$$

$$= \tilde{x}_{k+(\Delta l+\Delta p)} + \tilde{x}_{k-\Delta p} - (\tilde{x}_{k-\Delta p} + \tilde{x}_k - \tilde{z}_k)$$

$$= \tilde{x}_{k+(\Delta l+\Delta p)} + \tilde{x}_{k-\Delta p} - \tilde{z}_{k-\Delta p}$$

The transition marked with (*) uses the recursive form of the sequence derivation.

- The cases for $k+1$ to $k+(\Delta l+\Delta p)+1$, ..., and $k+(\Delta l+\Delta p)-1$ to $k+2(\Delta l+\Delta p)$ are similar to the one above and are therefore not shown explicitly. ❑

Relational Pattern of MaxOA

The relational operator pattern to implement the MaxOA derivation algorithm (figure 10) is based on the explicit form and can be seen as an extension of the pattern to reconstruct raw data values from cumulative queries (section 3.1). The inner query joining s_1 and s_2 corresponds to the two sums computing the compensation terms. Again, a CASE-statement is used to perform necessary negations yielding the correct operand for the SUM aggregation function in a subsequent grouping operation. The result of the inner query is joined again using a left outer join to preserve the original sequence values at the lower positions not exhibiting a compensation term from the inner query. Again, a CASE-statement is used for generating positive and negative terms.

4.2. General MaxOA Derivation Pattern

The MaxOA is introduced in the former subsection with the restriction of sharing the common upper window bound. This subsection extends this framework to derive a sequence $\tilde{y} = (l_y, h_y)$ from a sequence $\tilde{x} = (l_x, h_x)$ with the coverage factors $\Delta l = l_y - l_x > 0$ and $\Delta h = h_y - h_x > 0$. Using two single side derivation mechanisms, we give the general derivability algorithm:

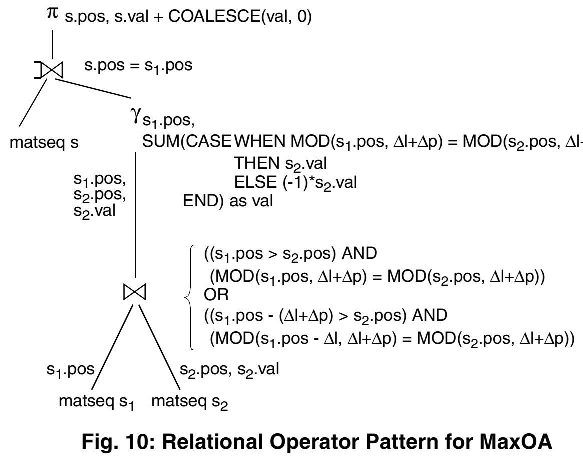
π s.pos, s.val + COALESCE(val, 0)
s.pos = s1.pos
γ s1.pos,
SUM(CASE WHEN MOD(s1.pos, Δl+Δp) = MOD(s2.pos, Δl+Δp))
THEN s2.val
ELSE (-1)*s2.val
END) as val
matseq s
s1.pos,
s2.pos,
s2.val
((s1.pos > s2.pos) AND
(MOD(s1.pos, Δl+Δp) = MOD(s2.pos, Δl+Δp))
OR
((s1.pos - (Δl+Δp) > s2.pos) AND
(MOD(s1.pos - Δl, Δl+Δp) = MOD(s2.pos, Δl+Δp))
s1.pos
s2.pos, s2.val
matseq s1
matseq s2

Fig. 10: Relational Operator Pattern for MaxOA

Theorem: *Double Side Sequence Derivability*

A sequence $\tilde{y} = (l_y, h_y)$ is derivable from a sequence $\tilde{x} = (l_x, h_x)$ by applying the single side derivation pattern twice:

$$\tilde{y}_k := \tilde{x}_k + (\tilde{x}_{k-\Delta l} - z_k^L) + (\tilde{x}_{k+\Delta h} - \tilde{z}_k^H)$$

where $\tilde{z}_k^L$ and $\tilde{z}_k^H$ are the compensation sequences defined as:

$$\tilde{z}_k^L := \tilde{x}_{k-\Delta l} - \tilde{x}_{k-(\Delta l+\Delta p)} + \tilde{z}_{k-\Delta p}^L$$

$$\tilde{z}_k^H := \tilde{x}_{k+\Delta h} - \tilde{x}_{k+(\Delta l+\Delta q)} + \tilde{z}_{k+\Delta q}^H$$

The coverage and overlap factors are then determined by:

$$\Delta l = l_y - l_x > 0 \text{ and } \Delta h = h_y - h_x > 0$$

$$\Delta p = 1 + h_x + l_x - \Delta l \text{ and } \Delta q = 1 + l_x + h_x - \Delta h$$

The explicit formula is constructed following the same way.

$$\tilde{y}_k := \tilde{x}_k + \sum_{i=1}^{\infty} \tilde{x}_{k-i(\Delta l+\Delta p)} - \sum_{i=1}^{\infty} \tilde{x}_{k-((i+1)\Delta l+i\Delta p)} + \sum_{i=1}^{\infty} \tilde{x}_{k-i(\Delta h+\Delta q)} - \sum_{i=1}^{\infty} \tilde{x}_{k-((i-1)\Delta h+i\Delta q)}$$

Clarification: Instead of giving a complete proof, we refer the reader to figure 11 extending figure 9 with regard to the general derivation pattern.

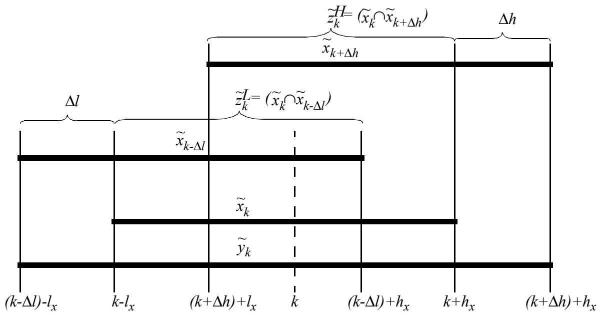

Fig. 11: Maximal Overlaping Double Side Variant

Obviously, the relational mapping of the general derivation pattern can be easily adapted from the restricted case, discussed in the preceding subsection. We may add here that – in comparison to MinOA – MaxOA can be used to handle materialized sequence views defined over MIN/MAX. It is easy to see that $\tilde{y}_k := min(\tilde{x}_{k-\Delta l}, \tilde{x}_{k+\Delta h})$ in the general case.

5. The MinO Algorithm

Another alternative for deriving sequence queries from materialized sequence data consists in the approach of considering a relationship with minimal overlapping. In contrast to the method presented in the preceding section, the proposed MinOA ("*Minimal Overlapping Algorithm*") results in easier specification, but does not allow the derivation of MIN/MAX sequences. Again, we consider the SUM derivation of a sliding window sequence $\tilde{y}=(l_y,h_y)$ from a materialized sequence $\tilde{x}=(l_x,h_x)$ with coverage factors $\Delta l=l_y-l_x$ and $\Delta h=h_y-h_x$.

Basic Derivation Pattern

The MinOA may be seen as an direct extension of the algorithm reconstructing single data points from a materialized sequence of windows (section 3.1). Two complete sequences are constructively generated so that their difference returns exactly the desired sequence values $\tilde{y}_k$ (figure 12):

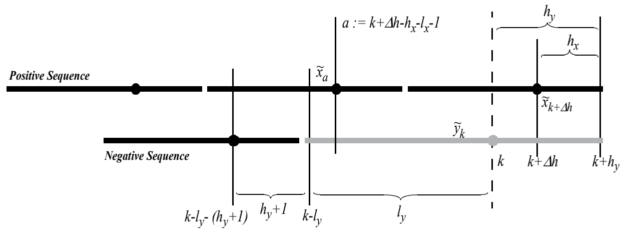

Fig. 12: Minimal Overlap Algorithm

- *Positive Sequence:* The head of the positive sequence is adjusted so that it is right justified with $\tilde{y}_k$. Therefore the center of this head is $k+\Delta h$, i.e. it is designated by $\tilde{x}_{k+\Delta h}$. The remaining elements of the positive sequence are computed by shifting the center to the left by multiples of W_x. The number of shifts is finite because $\tilde{x}_k$ will be zero if $k \leq -h_x$.
- *Negative Sequence:* The negative sequence has to fill the gap between the origin and $\tilde{y}_k$. So the center of the head of this sequence is found by going backwards from k to the beginning of $\tilde{y}_k$ ($-h_y$) and additionally h_x+1 to come from the upper bound to the center resulting in $\tilde{x}_{k-(l_y+h_y+1)}$. The remaining elements of the sequence are again generated by left shifting in multiples of the window size W_x. Again this sequence is finite.

Theorem: *Sequence Derivation (MinOA)*
A sequence $\tilde{y}=(l_y,h_y)$ is derivable from a complete sequence $\tilde{x}=(l_x, h_x)$ by the following explicit formula:

$$\tilde{y}_k := \sum_{i=0}^{\infty} \tilde{x}_{k+\Delta h-iw_x} - \sum_{i=1}^{\infty} \tilde{x}_{k-\Delta l-iw_x}$$

Clarification: We intentionally omit a formal proof here because of the constructive way of affiliating the formulas. However, it might be worth mentioning here that the upper bound of the summations in the explicit form can be limited as well by a parameter i_{up}, which is the minimal i fulfilling the following condition:

$$x_{k+\Delta h-i_{up}w_x} = 0 \Leftarrow k+\Delta h-i_{up}w_x \leq -h:\ i_{up} \geq \frac{k+h_y}{w_x}$$

For example i_{up} can be set to: $i_{up} = \left\lceil \frac{k+h_y}{w_x} \right\rceil$

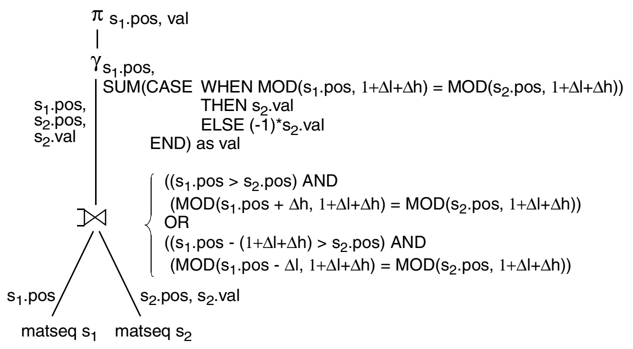
π s1.pos, val
γ s1.pos,
SUM(CASE WHEN MOD(s1.pos, 1+Δl+Δh) = MOD(s2.pos, 1+Δl+Δh))
THEN s2.val
ELSE (-1)*s2.val
END) as val
s1.pos,
s2.pos,
s2.val
((s1.pos > s2.pos) AND
(MOD(s1.pos + Δh, 1+Δl+Δh) = MOD(s2.pos, 1+Δl+Δh))
OR
((s1.pos - (1+Δl+Δh) > s2.pos) AND
(MOD(s1.pos - Δl, 1+Δl+Δh) = MOD(s2.pos, 1+Δl+Δh))
s1.pos
s2.pos, s2.val
matseq s1
matseq s2

Fig. 13: Relational Operator Pattern for MinOA

Relational Pattern of MinOA

The relational pattern to implement the MinOA approach is again adapted from the pattern deriving single raw data values from a materialized sequence view (section 2.2). In comparison to MaxOA, the MinOA does not need a second join with the materialized view, because the explicit form given above does not exhibit a third operand. However, a left outer join is used again, to prevent the elimination of the first sequence values.

6. Reporting Function Sequences

As illustrated in the introduction, reporting functions allow the optional specification of a complex ordering and partitioning scheme. This section emphasizes these perspectives from the derivability point of view. In a preparing step, working with a linear ordering over multiple columns requires a positioning function returning the global position of a multidimensional sequence entry. Moreover, we finally extend the notion of simple sequences to reporting sequences formally reflecting the power of reporting functions in SQL.

Definition: *Position Function*
A position function $pos: N^n \rightarrow N$ returns a global position k according to the linear ordering of the list of the parameters $k_1,...,k_n$, i.e. $pos(k_1,...,k_n) = k$.
For $n = 1$, the position function is equivalent to $id()$.

Definition: *Reporting Sequences*
A reporting sequence is a simple sequence extended by a partitioning scheme consisting of a set of partitioning attributes and an ordering scheme consisting of a list of ordering columns $k_1,...,k_n$.

6.1. Ordering Reduction

A first direction when considering deriving reporting sequences ordered according to multiple columns occurs if a sequence $\tilde{y}$ has to be derived from a sequence $\tilde{x}$ by reducing the number of ordering columns $(k_1,...,k_n)$ to $(k_1,...,k_{n-j})$. Reducing the number of ordering columns implies some aggregation semantics, because sequence data which is no longer ordered wrt. a column k_i is invariant wrt. that column and sequence values identified by different k_i-values have to be collapsed into a single sequence value. To perform the collapsing process of $\tilde{x}$-values into $\tilde{y}$-values within the given framework of derivability, we require that the prefix of the linear ordering scheme remains preserved or – in other words – the list of ordering columns may be reduced from right to the left starting with k_n and keeping at least k_1.

Lemma: *Derivation of Reporting Sequences by Ordering Reduction*

A reporting function $\tilde{y}$ is derivable from a reporting function $\tilde{x}$, $\tilde{y} \prec \tilde{x}$, by preserving a prefix of the ordering scheme of $\tilde{x}$, i.e.

$$\tilde{y}(k_1,...,k_{n-j}) \prec \tilde{x}(k_1, ...,k_{n-j},1,...,1)$$

The window specification of $\tilde{y}$ results dependent on the global position k in:

$$w'_L(k) = k - pos((k_1,...,k_{n-j}) - 1, 1, ...,1)$$
$$w'_H(k) = pos((k_1,...,k_{n-j})+1, 1, ...,1) - k - 1$$

The basic idea of this derivation approach is that the 'first' sequence entry of $\tilde{y}$ with regard to the remaining ordering columns is derived from $\tilde{x}$. To achieve the correct results with overlapping windows, we refer to the new window specification and use the same derivation process as in the case of simple sequences, i.e. MinOA or MaxOA. The specification of the window boundaries makes heavy use of the *pos()*-function when computing the next lower or upper address wrt. the remaining ordering columns. For example, suppose the right most ordering column of the three-column sequence address *(2,4,2)* has to be eliminated (i.e. $j=1$). The lower bound of the corresponding window is then evaluated by

$$w_L(k) = k - pos((2,4)-1, 1) =$$
$$= k - pos((2,3) , 1) = k - pos(2,3,1),$$

and analogously the upper bound yields

$$w_H(k) = k - pos((2,4)+1, 1) =$$
$$= k - pos((3,1) , 1) = k - pos(3,1,1).$$

6.2. Partitioning Reduction

Within a derivation process, not only the number of ordering columns but also the number of partitioning columns may be reduced. In general, the partitioning mechanism defines the restart conditions for the proposed sequence specification. An example is shown in the introduction where the cumulative transaction sum is computed on a total and on a monthly basis. As seen in section 3.2, additional information (header/trailer) is needed at the beginning of a sequence to accomplish a correct derivation process. Therefore, we may learn that a sequence has to provide a header/trailer for each partition. This leads to the notion of a complete reporting function and specification of derivation requirements in the case of partitioning reduction.

Definition: *Complete Reporting Function*

A reporting function is complete if it provides header/trailer information for each partition resulting from the given partitioning scheme.

Lemma: *Derivation of Reporting Sequences by Partitioning Reduction*

A reporting function $\tilde{y}$ is derivable by partitioning reduction from a reporting function $\tilde{x}$, if $\tilde{x}$ is complete.

7. Evaluation Considerations

The presented algorithms to compute reporting functions and derive reporting function queries from materialized views were implemented in SQL and tested using IBM DB2V7.1 running on a PII-466 Linux machine.

Table 1 illustrates measured query runtimes to compare execution times for computing sequence values from raw data. As we can see and as we might have expected, the use of existing reporting functionality inside the database engine is always beneficial. However, we can observe that the performance of the self join method to simulate reporting functionality heavily relies on the existence of an index on the sequence position. Query execution time is then roughly cut down by 95% and is approximately double the time needed with the support of reporting functionality.

# seq values	no primary index		with primary key index	
	reporting func.	self join method	reporting functionality	self join method
5.000	0.751s	39.016s	0.701s	1.822s
10.000	1.482s	157.656s	1.492s	3.675s
15.000	2.244s	357.774s	2.284s	5.528s

Table 1: Computing Sequence Data

In opposite to the complete recomputation, we compare MinOA and MaxOA in a second scenario (including primary key indexes). As we can see in table 2, we considered four alternative implementations. The MaxOA as well as the MinOA were implemented with a disjunctive join predicate and as a union over queries with simple predicates. We may observe from the experimental runs that the execution of disjunctive join predicates is far more beneficial than splitting the query and computing each partition for itself. Moreover, we can see that the overall query runtimes behave not in a linear manner, so that the approach of reconstructing sequence values in a relational way without any other special operators is not advisable for large sequences.

# seq values	MaxOAlgorithm		MinO Algorithm	
	disjunctive predicate	union of simple pred. queries	disjunctive predicate	union of simple pred. queries
100	0,184	0,650	0,288	0,479
500	3,290	7,800	6,401	6,253

Table 2: Computing Sequence Data

# seq values	MaxOAlgorithm		MinO Algorithm	
	disjunctive predicate	union of simple pred. queries	disjunctive predicate	union of simple pred. queries
1000	12,819	35,883	25,137	28,023
1500	28,621	81,995	55,823	63,691
2000	50,663	149,223	99,598	120,739
3000	727,998	542,216	576,296	272,575
5000	2063,054	1561,459	1635,215	765,280

Table 2: Computing Sequence Data

To put it into a nutshell, we may conclude from the experiments that simulation of reporting functionality is feasible as soon as we can rely on the existence of an index structure which can be exploited during join processing. From the derivation process, we can see that working with a disjunctive join predicate is more beneficial compared to two independent queries with simple predicates. The comparison of MinOA and MaxOA does not produce a real winner. Although from a theoretical perspective MinOA should be more economical, MaxOA is applicable to a broader spectrum of aggregation functions, especially with regard to MIN() and MAX() aggregation functions.

8. Summary and Conclusion

Within this paper we address the topic of sequence data processing in a data warehouse environment. Inspired by many applications in the warehouse context, sequence processing is attracting more and more attention in the relational database community. The introduction of reporting functions as the SQL extension underlines this importance. However, this functionality needs to be incorporated into the database technology especially used within the focussed application area. This is certainly not an easy but time and resource consuming task. Using our work, reporting function processing in the context of materialized views can be achieved without extending the internal structure of the database engine.

After an informal introduction of reporting functions in SQL and formally introducing the concept of a sequence defined using the OVER()-clause, the paper focusses on the problem of deriving sequence queries from materialized sequence data. We propose two alternative algorithms MaxOA and MinOA and give a relational representation without any specific sequence functionality required. The paper closes with a short comparative evaluation of these algorithms. Although this paper addresses some problems arising in the context of evaluating sequence queries in a set-oriented relational data and processing model, many other issues remain open. In summary, we think that sequence processing is a tremendously interesting area with a huge potential for research as well as for commercial development.

References

[1] Agrawal, S.; Agrawal, R.; Deshpande, P. M.; Gupta, A.; Naughton, J.F.; Ramakrishnan, R.; Sarawagi, S.: On the Computation of Multidimensional Aggregates. In: *VLDB'96*, pp. 506-521

[2] Baralis, E.; Paraboschi, S.; Teniente, E.: Materialized Views Selection in a Multidimensional Database. In: *VLDB'97*, pp. 156-165

[3] Chaudhury, S.; Krishnamurthy, R.; Potamianos, S.; Shim, K.: Optimizing Queries with Materialized Views. In: *ICDE'95*, pp. 190-200

[4] Deshpande, P.M.; Ramasamy, K.; Shukla, A.; Naughton, J.F.: Caching Multidimensional Queries Using Chunks. In: *SIGMOD'98*, pp. 259-270

[5] Gray, J.; Bosworth, A.; Layman, A.; Pirahesh, H.: Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total. In: *ICDE'96*, pp. 152-159

[6] Gupta, H.; Harinarayan, V.; Rajaraman, A.; Ullman, J.D.: Index Selection for OLAP. In: *ICDE'97*, pp. 208-219

[7] Gunopulos, D.; Das, G.: Time Series Similarity Measures. Tutorial *SIGMOD2001*

[8] Gupta, A.; Mumick, I. S.: *Materialized Views: Techniques, Implementations, and Applications*. Cambridge (Massacusetts), London (England), 1999

[9] Harinarayan, V.; Rajaraman, A.; Ullman, J.D.: Implementing Data Cubes Efficiently. In: *SIGMOD'96*, pp. 205-216

[10] N.N.: SQL Reference: OLAP Functions. IBM Corp., 2001 (http://www-4.ibm.com/cgi-bin/db2www/data/db2/udb/winos2unix/support/v7pubs.d2w/en_main)

[11] Lehner, W.; Cochrane, R.; Pirahesh, H.; Zaharioudakis, M.: fAST Refresh using Mass Query Optimization. In: *ICDE2001*, pp. 391-398

[12] Haas, P.J.; Naughton, J.F.; Seshadri, S.; Stokes, L.: Sampling-Based Estimation of the Number of Distinct Values of an Attribute. In: *VLDB'95*, pp. 311-322

[13] N.N.: Oracle 9i SQL Reference Manual. Oracle Corp., 2001 (http://technet.oracle.com)

[14] Sarawagi, S.: Indexing OLAP data. In: *IEEE Data Engineering Bulletin 20(1997)1*, pp. 36-43

[15] Scheuermann, P.; Shim, J.; Vingralek, R.: WATCHMAN: A Data Warehouse Intelligent Cache Manager. In: *VLDB'96*, pp. 51-62

[16] Seshadri, P.; Livny, M.; Ramakrishnan, R.: SEQ: A Model for Sequence Databases. In: *ICDE'95*, pp. 232-239

[17] Winter, R.: SQL-99's New OLAP Functions. *Intelligent Enterprise*, 3(2000)2

[18] Yang, J.; Karlapalem, K.; Li, Q.: Algorithms for Materialized View Design in Data Warehousing Environment. In: *VLDB'97*, pp. 136-145

[19] Yi, B.-K.; Faloutsos, C: Fast Time Sequence Indexing for Arbitrary Lp Norms. In: *VLDB2000*: pp. 385-394

[20] Zachariodakis, M.; Cochrane, R.; Pirahesh, H.; Lapis, G.; Urata, M.: Answering Complex SQL Queries Using Summary Tables. In: *SIGMOD2000*, pp. 105-116

[21] Zhao, Y.; Deshpande, P.M.; Naughton, J.F.; Shukla, A.: Simultaneous Optimization and Evaluation of Multiple Dimensionale Queries. In: *SIGMOD'98*, pp. 271-282