Analyzing Sequential Data in Standard OLAP Architectures

Christian Koncilia[1], Johann Eder[1], and Tadeusz Morzy[2]

[1] Alpen-Adria-Universität Klagenfurt
Dep. of Informatics-Systems
{eder,koncilia}@isys.uni-klu.ac.at
[2] Poznan University of Technology
Institute of Computing Science
morzy@put.poznan.pl

Abstract. Although nearly all data warehouses store sequential data, i.e. data with a logical or temporal ordering, traditional data warehouse or OLAP approaches fail when it comes to analyze those sequences. In this paper we will present a novel approach which generates query-specific subcubes, i.e. subcubes that consist only of data which fulfill a given sequential query pattern. These subcubes may then be analyzed using standard OLAP tools. Our approach consists of two functions which both return such subcubes. Hence, the user can still use all the well-known OLAP operations like drill-down, roll-up, slice, etc. to analyze the cube. Furthermore, this approach may be applied to all data warehousing architectures.

1 Introduction

Business Intelligence (BI), Data Warehousing (DWH), and On-Line Analyical Processing (OLAP) enable users to perfomantly analyze mass data by storing data in No-SQL database systems, e.g. multidimensional database systems, or by applying DWH specific logical schemas to relational database systems, e.g. the Star Schema, Snowflake Schema, etc. [6].

Traditional business intelligence tools analyze facts along dimensions. Facts describe what a user wants to analyze whereas dimensions describe how the user analyses his data [6]. Typical examples for facts are Turnover, Profit, the Stock of Inventory, etc. These facts may then be analyzed along a set of dimensions like Time, Products or Geography.

This approach succeeded to proof its feasibility in innumerable implementations in many industrial sectors. However, this approach fails when it comes to efficiently analyze sequential data, i.e. data with a logical or temporal ordering [9].

Why does the traditional DWH approach fail when it comes to sequential data analysis? Assume that we store data about treatment costs and diagnoses for patients in a DWH. Traditional data warehouses are built to answer questions like "what are the total costs for patients in 2010" or "what are the average costs for all patients diagnosed cerebral infarction". However, they are not prepared to answer queries like "what are the follow-up costs of patients diagnosed cerebral infarction within 12 months after the diagnose". This even gets more complicated, when analyzing data along several events, e.g. when analyzing follow-up costs for patients with a certain diagnose who received a certain treatment within a given time period after the diagnose.

Although sequential data representation is not a new research area, the fact that most data sets in an OLAP system are sequential by nature has been ignored until recently, e.g. in [1,9,8]. These approaches focus on developing novel data warehouse / OLAP architectures. This allows to develop new operators, query languages, indexing and caching strategies, etc. However, in our opinion there is also an evident need to analyze sequential data in existing OLAP infrastructures.

Contribution: In this paper we will present a sophisticated approach which enables the user to analyze simple atomic events and complex sequences of events. In contrast to other approaches (which will be discussed in section 7), our approach smoothly integrates into a standard OLAP architecture. Basically, our approach consists of the following steps:

1. The user defines the sequence she / he wants to analyze, e.g. all patients who had a specific diagnose A after a diagnose B.
2. A subcube is generated which contains all relevant data, e.g. all patients records for all patients who had a diagnose A after a diagnose B.
3. An additional dimension Relative Time Axis is created enabling the user to analyze data in a very flexible way.

The result of a sequential query in our approach is itself a standard OLAP (sub-)cube. Hence, the user can still use all the well-known OLAP operations like *drill-down*, *roll-up*, *slice*, *dice* and so on to analyze this cube.

This paper is organized as follows: In section 2 we will briefly describe a motivating example which we will use throughout the rest of this paper to depict the application of our approach. Section 3 will provide a formal model of a data warehouse which we will extend in section 4 with our sequential OLAP approach. We will present the prototypical implementation of our approach in section 5. In section 6, we are going to briefly discuss some application areas for a sequential OLAP approach. Related work will be discussed in section 7. Finally, we will conclude this paper in chapter 8.

2 Motivating Example

In this section, we will present our motivating example which we will use as running example throughout the rest of the paper. Consider a database with the following table storing information about patients, diagnoses and treatment costs:

Patient	Diag	Date	Costs
Tim	I26	1-1	50
Tim	C11	1-2	70
Walter	I26	1-8	45
Tim	I27	1-8	110
John	B32	1-2	80
Walter	C11	1-2	60

In this example the patient Tim went to a doctor on 1/1/10 and was diagnosed with ICD (International Classification of Diseases) code I26 (the code for the disease *pulmonary embolism*). The next day, he wanted to get a second opinion and went to a different doctor who diagnosed a different disease encoded C11. Then, a few days later, he went to a third doctor who diagnosed I27.

The star schema for a data warehouse to analyze this information consists of a fact table storing the costs, and three dimension tables (*Patient*, *Diagnose*, *Date*). Easily one can use this data warehouse to answer queries like "what are the total costs for patient Tim in 2010" or "what are the average costs for all patients diagnosed I26". However, such a data warehouse structure would not be suitable to answer queries where a dimension member depends on another dimension member, i.e. where we have sequences.

As we will discuss in section 4.1, such a sequential OLAP query may be based on *Atomic Sequences* or on *Complex Sequences*. Queries based on atomic sequences are queries that make use of only one single event, e.g. "What are the follow-up costs of patients during three month after she/he has been diagnosed I26". In this example, the single event would be the diagnose I26.

In contrast to such a query, a query based on complex sequences consists of two or more events. An example for such a query would be: "How many patients have been diagnosed H35 (Retinopathie, a disease often caused by diabetes which can lead to blindness) within 12 months after they have been diagnosed E10 (diabetes)." This query would consist of two events, namely the diagnose E10 and the diagnose H35.

Of course, such a complex sequence query is not restricted to two events nor is it restricted to events that stem from one dimension in the data warehouse. For instance, if we would store prescriptions in our data warehouse we could also state queries like: "What are the average follow-up costs of a diagnose I26 for patients that have been prescribed Heparin within 6 months after the diagnose". This query would consist of two events stemming from two different dimensions.

3 Formal OLAP Model

In this section we will give a formal definition of a data warehouse based on the model presented in [7]. Later on we will extend this data warehouse model such that the user is able to state all kinds of sequential queries. Please note that our approach for sequential OLAP simply extends the standard OLAP approach.

The result of a sequential query is itself a standard (sub-)cube, extended with a set of relative time axes. Hence, the user can still use all the well-known OLAP operations like drill-down, roll-up, slice, dice and so on to analyze her or his cube.

Intuitively, we define the schema of a data warehouse as a set of cubes which again are defined as a set of dimensions. The schema of each dimension is defined by a set of categories, e.g., the dimension *Date* might consist of the categories *Year*, *Month* and *Day* organized in a hierarchical relation $Year \rightarrow Month \rightarrow Day$, where for example $Year \rightarrow Month$ means that a month rolls-up to a year.

Each category consists of a set of dimension members. Dimension members define the instances of a data warehouse schema. For instance, *January*, *February* and *March* are dimension members assigned to the category *Month*.

Formally, the schema of a data warehouse is defined by:

i.) A number of dimensions J.
ii.) A set of dimensions $\mathbb{D} = \{D_1, ..., D_J\}$, where $D_i =< ID, D_{Key} >$. ID is a unique identifier of the dimension. D_{Key} is a user defined key (e. g. , the name of the dimension), which is unique within the data warehouse.
iii.) A number of categories K.
iv.) A set of categories $\mathbb{C} = \{C_1, ..., C_K\}$ where $C_i =< ID, C_{Key} >$. ID is a unique identifier of the category. C_{Key} is a user defined key (e. g. , the name of the category) which is unique within the data warehouse.
v.) A set of assignments between dimensions and categories $\mathbb{A}_{DC} = \{A^1_{DC}$, ..., $A^N_{DC}\}$, where $A^i_{DC} =< D.ID$, $C.ID >$. $D.ID$ represents the identifier of the corresponding dimension. $C.ID$ represents the identifier of the corresponding category.
vi.) A number of hierarchical category assignments O.
vii.) A set of hierarchical category assignments $\mathbb{HC} = \{HC_1, ..., HC_O\}$ where $HC_i =< ID, C.ID_C, C.ID_P >$. ID is a unique identifier of the hierarchical category assignment. $C.ID_C$ is the identifier of a category, $C.ID_P$ is the category identifier of the parent of $C.ID_C$ or $\emptyset$ if the category is a top-level category.
viii.) A number of cubes I.
ix.) A set of cubes $\mathbb{B} = \{B_1, ..., B_I\}$ where $B_i =<ID$, B_{Key}, $\mathcal{S} >$. ID is a unique identifier of the cube (similar to $O_{id's}$ in object-oriented database systems). B_{Key} is a user defined key (e. g. , the name of the cube), which is unique within the data warehouse.
$\mathcal{S}$ represents the schema of the cube. The tuple $\mathcal{S}$ consists of all dimensions and hierarchical category assignments that are a part of this cube. Therefore, $\mathcal{S}$ is defined as $\mathcal{S} = (\mathbb{D}, \mathbb{A})$ where $\mathbb{D} = \{D_1.ID, ..., D_N.ID\}$ $(N \leq J)$ and $\mathbb{A} = \{HC_1.ID, ..., HC_M.ID\}$ $(M \leq O)$.

The instances of a data warehouse are defined by:

i.) A number of dimension members P.
ii.) A set of dimension members $\mathbb{M} = \{M_1, ..., M_P\}$ where $M_i =< ID$, M_{Key}, $\mathbb{CA}$, $>$. ID is a unique identifier of the dimension member. M_{Key} is a user

defined key (e. g. , the name of the dimension member), which is unique within the data warehouse. The set $\mathbb{CA}$ represents the set of categories, to which the corresponding dimension member is assigned.

iii.) A set of hierarchical member assignments $\mathbb{HM} = \{ HM_1, ..., HM_O\}$ where $HM_i =< ID, M.ID_C, M.ID_P, f >$. ID is a unique identifier of the hierarchical member assignment. $M.ID_C$ is the identifier of a dimension member, $M.ID_P$ is the dimension member identifier of the parent of $M.ID_C$ or $\emptyset$ if the dimension member is at the top-level. f represents the consolidation function between $M.ID_C$ and $M.ID_P$, e. g. $+$ for addition, $-$ for subtraction, etc.

iv.) A function $cval : (M_{D_1}, ..., M_{D_N}) \to measure$, which uniquely assigns a measure to each vector $(M_{D_1}, ..., M_{D_N})$ where $(M_{D_1}, ..., M_{D_N}) \in \mathcal{M}_{D_1} \times ... \times \mathcal{M}_{D_N}$. The domain of this function is the set of all cell references. The range of this function are all measures of a cube.

4 Sequential OLAP Model

In this chapter, we will extend the OLAP model presented in section 3. The extension basically consists of two items: 1) we will introduce the concept of sequential OLAP functions and 2) we will enrich this model with the concept and definition of a relative time axis.

Intuitively, a sequential OLAP function can be considered as an extended slice operation and a relative time axis represents the time difference between a given event and any other event.

4.1 Sequential OLAP Function and Events

Basically, a sequential OLAP function takes a cube, a grouping dimension, an ordering dimension and a sequence of events as input and returns a subcube as output. The terms *grouping dimension*, *ordering dimension* and *sequence of events* will be defined in section 4.2. The query "fetch all patient records for patients which have been diagnosed retinopathie after they have been diagnosed diabetes" could be an example for a sequential OLAP function. This query would result in a subcube that consists of all dimensions of the corresponding cube and all dimension members and measures which belong to patients that have been diagnosed retinopathie after a diagnose diabetes. This subcube may then serve as basis for analysis which for instance easily enable the user to compute follow-up costs.

The fundamental basis for our sequential OLAP function are sequences. We distinguish between two different kinds of sequences:

1.) **Complex Sequence:** A complex sequence forms a path through a set of events, e.g. a sequence $\mathcal{E}_1 \to \mathcal{E}_2 \to \ldots \to \mathcal{E}_n$ where $\mathcal{E}_i$ is an event.

2.) **Atomic Sequence:** An atomic sequence is a subset of complex sequence. It represents a one stepped path, i.e. $\mathcal{E}_1$. For instance, $\mathcal{E}_1$ may be the event "diagnosed diabetes".

An event $\mathcal{E}$ is an appearance of an incident at a given point of time. In our context, we can define an event $\mathcal{E}$ as the existence of a function *cval* with $cval(M_t, M_e, \ldots) \neq null$ with a given dimension member M_e that defines the incident and a dimension member M_t that defines the point of time.

$\mathcal{E}_j \rightarrow \mathcal{E}_k$ means that the event $\mathcal{E}_k$ occurred directly after event $\mathcal{E}_j$, i.e. there exists no event $\mathcal{E}_l$ between $\mathcal{E}_j$ and $\mathcal{E}_k$ along an ordering dimension defined by the user. Usually, this ordering dimension will be the time dimension.

4.2 Sequential OLAP Function for Atomic Sequences

As a complex sequence can be decomposed to a set of atomic sequences, we will start by defining the sequential OLAP function for atomic events.

Pre-Conditions: The following pre-conditions for a sequential OLAP function on atomic sequences have to be fulfilled:

1.) A cube B_i has to be defined as in section 3. This cube serves as input, i.e. it defines the base for the sequential OLAP function.
2.) B_i has to contain at least one ordering dimension D_o. An ordering dimension is a dimension on which an ordering function f_{order} has been defined. $f_{order}(M_j, M_k)$ takes any two dimension members M_j and M_k and returns -1 if $M_j < M_k$, 0 if $M_j = M_k$ or $+1$ if $M_j > M_k$.
3.) The user has to define a grouping dimension D_g. This grouping dimension defines the subject of the analysis. Hence, D_g defines which dimension the event refers to, i.e. which dimension the order of the ordering dimension refers to. D_g may be any dimension of B_i.
4.) Furthermore, the user has to define a single (atomic) event $\mathcal{E}$ with $\mathcal{E} = M_E$ where M_E is a dimension member of D_E and D_E is a dimension in B_i.

Definition: Now, the function *solap* can be defined as follows: Given the input B_i, D_o, D_g and $\mathcal{E}$ the function $solap(B_i, D_o, D_g, \mathcal{E}, \mathcal{E}_p)$ returns a subcube B_o which consists of all dimensions $D_i \in B_i$ and all dimension members M_i with $M_g \in D_g \wedge \exists cval(M_{D_1}, \ldots, M_i, M_g, M_E, \ldots, M_{D_N}) \neq null \wedge M_E = \mathcal{E}$.

Please note that $\mathcal{E}_p$ is not used in atomic sequences and will be discussed later on in section 4.3.

Intuitively we can say that a $solap(B_i, D_o, D_g, \mathcal{E}, \mathcal{E}_p)$ returns a subcube which consists of all the data of all dimension members in the grouping dimension for which there exists at least one entry in the fact table that represents the given event.

Example: Assume that B_i is the cube as defined in our running example in section 2. The ordering dimension D_o is the dimension *Date*. The grouping dimension D_g, i.e. the subject of our analysis, is the dimension *Patient*. The event $E = I26$.

Taking these input parameters, the function $solap(B_i$, *Date*, *Patient*, $I26)$ would return a subcube which consists of all the data of all patients who had a diagnose I26, i.e. it would return a subcube which consists of the data represented in the following table:

Patient	Diag	Date	Costs
Tim	I26	1-1	50
Tim	C11	1-2	70
Walter	I26	1-8	45
Tim	I27	1-8	110
Walter	C11	1-2	60

4.3 Sequential OLAP Function for Complex Sequences

In the previous section we defined the function *solap* for atomic events. We will now extend this function to work on complex sequences.

Pre-Conditions: The pre-conditions are the same as defined in section 4.2 except the fact that the user may define any sequence of events $\mathbb{E} =< \mathcal{E}_1, \ldots, \mathcal{E}_n >$ with $\mathcal{E}_i = M_{E_i}$ where M_{E_i} is a dimension member of D_{E_i} and D_{E_i} is a dimension in B_i.

Furthermore, as complex sequences have to consider the ordering of several events, we have to extend the *solap* function with an additional parameter, namely $\mathbb{E}_p$. In an atomic sequence, $\mathbb{E}_p$ is always *null*. In a complex sequence, $\mathbb{E}_p$ is the previous event in the sequence of events or *null*, if no previous event has been defined, i.e. if applying *solap* to the first event in a sequence of events.

Definition: First, extending the definition given in 4.2 with the parameter $\mathcal{E}_p$, the function *solap* can be defined as follows: Given the input B_i, D_o, D_g, $\mathcal{E}$ and $\mathcal{E}_p$ the function $solap(B_i, D_o, D_g, \mathcal{E}, \mathcal{E}_p)$ returns a subcube B_o which consists of all dimensions $D_i \in B_i$ and all dimension members M_i with $M_g \in D_g \wedge \exists cval(M_{D_1}, \ldots, M_i, M_g, M_E, \ldots, M_{D_N}) \neq null \wedge M_E = \mathcal{E} \wedge$
$\exists cval(M_{D_1}, \ldots, M_i, M_g, M_{E_p}, \ldots, M_{D_N}) \neq null \wedge M_{E_p} = \mathcal{E}_p \wedge f_{order}(\mathcal{E}, \mathcal{E}_p) > 0$.

Secondly, with the extended definition of the *solap* function, we can define *solap* for complex sequences: Given the input B_i, D_o, D_g and $\mathbb{E}$ the function $solap(B_i, D_o, D_g, \mathbb{E}, \mathbb{E}_p)$ can now be defined as a composition of *solap* functions on atomic sequences:

$$\begin{aligned} solap(B_i, D_o, D_g, \mathbb{E}) = \\ solap(solap(\ldots solap(B_i, D_o, D_g, \mathcal{E}_1, null) \ldots, \\ D_o, D_g, \mathcal{E}_{n-1}, \mathcal{E}_{n-2}), D_o, D_g, \mathcal{E}_n, \mathcal{E}_{n-1}) \end{aligned} \tag{1}$$

Example: Again, let B_i be the cube as defined in our running example in section 2, *Date* be the ordering dimension D_o and *Patient* be the grouping dimension D_g. Now, the user would like to analyze all patient records about patients who had a diagnose I26 and afterwards a diagnose I27. Hence, $\mathbb{E} =< I26, I27 >$.

Taking these input parameters, the function $solap(B_i$, *Date*, *Patient*, $< I26, I27 >)$ would result in a function $B_{o_1} = solap(B_i$, *Date*, *Patient*, $I26)$ whose result B_{o_1} would serve as input parameter for $B_{o_2} = solap(B_{o_1}$, *Date*, *Patient*, *I*27). Therefore, the resulting cube would consist of the data represented in the following table:

Patient	Diag	Date	Costs
Tim	I26	1-1	50
Tim	C11	1-2	70
Tim	I27	1-8	110

4.4 Relative Time Axis

The relative time axis function generates a new dimension in the cube which stores the difference between a given event and any other event. We will use the term relative time axis, although the concept of a relative time axis may be applied to any ordering dimension which doesn't necessarily have to be a time or date dimension.

In contrast to other time dimensions in the cube, the relative time axis is not a set of absolute timestamps like *12-30-2010* or *8-15-2010 10:42*, but a set of time intervals which are relative to the ordering dimension D_o (as described above, this ordering dimension is usually a time dimension). Thus, the relative time axis could for instance be a dimension with a set of dimension members $\{-n\ days, \ldots, -1\ day, 0, +1\ day, \ldots, +m\ days\}$.

Pre-Conditions: In order to compute a relative time axis, the following preconditions have to be fulfilled:

1.) A cube B_i has to be defined. Usually, this cube will be the result of a $solap()$ function as defined in sections 4.2 and 4.3.
2.) As defined in section 4.2 this cube B_i has to contain at least one ordering dimension D_o. Furthermore, the user has to define a grouping dimension D_g (the subject of the analysis) with $D_g \in B_i$.
3.) The user has to define a single event $\mathcal{E}$ with $\mathcal{E} = M_E$ where M_E is a dimenson member of D_E and D_E is a dimension in B_i.
4.) As there might exist several cell values in the cube referred to by a function $cval(M_1, \ldots, M_g, \mathcal{E}, \ldots, M_n)$ with M_g being a dimension member assigned to D_g, the user has to define which occurrence of $\mathcal{E}$ should serve as base. Currently, this can be done by applying a $first()$ or $last()$ function, which sets the first or last occurrence $\mathcal{E}$ as base. Other functions could be implemented.

Definition: we define a function $rta()$ (relative time axis) whish uses a function $diff()$ to compute the difference between any two event occurrences. $diff()$ takes two records, i.e. two $cval()$ functions as defined in section 3, and the ordering dimension D_o as input and computes the differences between the two entries. The granularity of $diff()$ is equal to the granularity of D_o, e.g. if the granularity of D_o is a day, then $diff()$ will return the difference in days.

The function $diff()$ may be defined by the user. Usually, it simply computes the difference between two dates:

$$diff(cval(M_{o_1}, M_g, M_E, \ldots), cval(M_{o_2}, M_g, \ldots)) =$$
$$M_{o_1} - M_{o_2}$$
$$with M_{o_1}, M_{o_2} \in D_o \wedge M_g \in D_g \wedge M_E \in \mathcal{E}. \quad (2)$$

Using the defined function $diff()$ we can formally define the $rta()$ function. $rta(B_i, D_o, \mathcal{E})$ returns a cube B_o where B_o consists of the same schema $\mathcal{S}$ as B_i and all dimension members $\mathbb{M}$, hierarchical member assignments $\mathbb{HM}$ and all measures assigned to B_i. Furthermore, B_o consists of an additional dimension D_{RTA} with a set of dimension members $\mathbb{M}_{RTA} = \{M_1, \ldots, M_n\}$ assigned to D_{RTA} (via $\mathbb{CA}$, $\mathbb{C}$ and $\mathbb{A}_{DC}$ as defined in section 3). For each $M_i \in \mathbb{M}_{RTA}$ we can define that $M_i.M_{Key} = diff(x, y)$ where $x = cval(M_{O_1}, \mathcal{E}, \ldots)$ and $y = cval(M_{O_2}, \ldots)$ and $x \neq y$.

Example: Assume that B_o is the resulting cube of the function $solap(B_i$, *Date*, *Patient*, $I26)$ as described in section 4.2. Again, the ordering dimension D_o is the dimension *Date*. The grouping dimension D_g, i.e. the subject of our analysis, is the dimension *Patient*. The event $E = I26$.

Taking these input parameters, the function $rta(B_o$, *Date*, *Patient*, $I26)$ would return a subcube which consists of all the data of all patients who had a diagnose I26. Furthermore, this subcube would consist of an additional dimension named RTA which stores the difference between the occurrence of a diagnose I26 and any other event. The following table depicts the resulting cube:

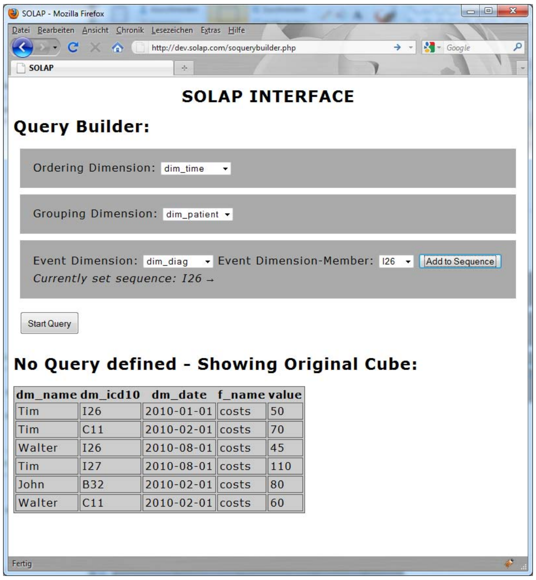
Patient	Diag	Date	Costs	RTA
Tim	I26	1-1	50	0
Tim	C11	1-2	70	+1
Tim	I27	1-8	110	+7
Walter	C11	1-2	60	-6
Walter	I26	1-8	45	0

4.5 Workflow Example

In this section we will discuss how a user may use SOLAP() and RTA() to state sequential OLAP queries and how she may analyse the resulting cube.

Assume that a user would like to state a query like “what are the follow-up costs for patients diagnosed I26 within 12 month after they have been diagnosed I26”? To answer this query the user would select the *Date* dimension as ordering dimension and *Patient* as grouping dimension. Furthermore, he defines an atomic sequence with one event “Diagnose = I26”. Now, the application would use the functions $SOLAP()$ and $RTA()$ (with the corresponding parameters) to generate a cube as depicted in Table 4.4.

This cube would enable the user to easily analyze the follow-up costs that occurred within 12 month after the diagnose I26. This could be done by applying standard OLAP functions to the cube. In this example, the user could simply


SOLAP - Mozilla Firefox
Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
http://dev.solap.com/soquerybuilder.php
SOLAP
Fertig

SOLAP INTERFACE

Query Builder:

Ordering Dimension: dim_time

Grouping Dimension: dim_patient

Event Dimension: dim_diag Event Dimension-Member: I26 Add to Sequence

Currently set sequence: I26 →

Start Query

No Query defined - Showing Original Cube:

dm_name	dm_icd10	dm_date	f_name	value
Tim	I26	2010-01-01	costs	50
Tim	C11	2010-02-01	costs	70
Walter	I26	2010-08-01	costs	45
Tim	I27	2010-08-01	costs	110
John	B32	2010-02-01	costs	80
Walter	C11	2010-02-01	costs	60

Fig. 1. Start Screen of our Prototype

select the dimension members $0 \ldots 12$ of the dimension RTA (which would correspond to a slice and dice operation) and calculate the sum of the fact *cost*. The same method could be applied to analyze which diagnoses occurred within 3 months before a diganose I26.

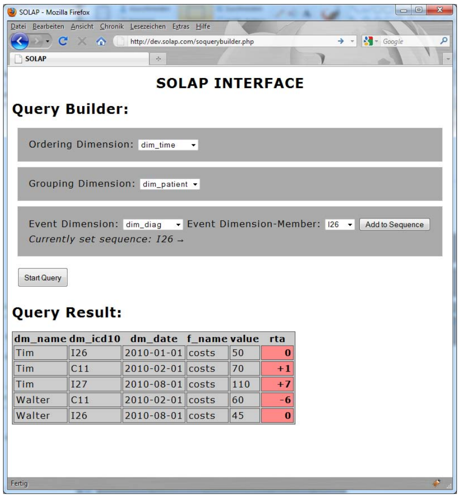
5 Proof of Concept

We implemented a prototype of our approach as proof of concept. This prototype has been implemented as a web-client using a PostgreSQL 9.0.0 database, PHP 5.3.2 and jQuery 1.4.2. Technically, the data warehouse itself has been built using the traditional Star Schema approach. Hence, we have one fact table and several tables representing the dimensions of the cube. For our running example, this results in a fact table that consists of the costs and foreign keys to the three dimensions: Patient, Diagnose and Date.

Figure 1 shows a screenshot of the start screen of the prototype. For this paper, we imported the data from our running example.

Using the prototype depicted in Fig. 2 the user may select an ordering dimension and a grouping dimension. Furthermore, she or he may define a sequence of events, i.e. an atomic sequence or a complex sequence. Currently, the prototype does not support using wildcards in sequences. In this example, the user selected a single event, i.e. Diagnose I26.

Basically, the application takes the user inputs, extracts the sequence defined by the user, and dynamically generates an SQL query for the first step in this


SOLAP - Mozilla Firefox
Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe
http://dev.solap.com/soquerybuilder.php
SOLAP
Add to Sequence
Start Query
Fertig

SOLAP INTERFACE

Query Builder:

Ordering Dimension: dim_time

Grouping Dimension: dim_patient

Event Dimension: dim_diag Event Dimension-Member: I26

Currently set sequence: I26 →

Query Result:

dm_name	dm_icd10	dm_date	f_name	value	rta
Tim	I26	2010-01-01	costs	50	0
Tim	C11	2010-02-01	costs	70	+1
Tim	I27	2010-08-01	costs	110	+7
Walter	C11	2010-02-01	costs	60	-6
Walter	I26	2010-08-01	costs	45	0

Fig. 2. Result Screen for an Atomic Sequence (Diagnose = I26)

sequence. This query serves as basis for a view created in the database. This view represents the subcube returned by the function $solap()$ as presented in section 4.2. For all subsequently defined sequence steps we repeate this process as defined in section 4.3. In contrast to the first step, all further steps work on the view defined in the previous step. Finally, the implementation calls the $rta()$ function as defined in section 4.4 to compute the relative time axis.

The result of this query is being depicted in figure 2. As can be seen, a new dimension "rta" has been created, representing the relative time axis.

6 Application Examples

In section 2 we discussed an application example originated in the health care sector. Basically, such a sequential OLAP approach would enrich each data warehouse that stores any kind of events, e.g. diagnoses, prescriptions, workflow tasks, sensor values and so on.

In this section we would like to briefly discuss some application examples for sequential OLAP are:

1.) **Workflow Systems:** Usually, a workflow system consists of several tasks. These tasks are linked with control structures like conditional branches, loops, joins and so on [13]. Analyzing worklow instances with OLAP or data warehouse techniques is tedious and sometimes impossible because of these control structures [5]. However, applying our sequential OLAP technique would enable us to reduce the complexity of an unlimited amount of possible instance structures to a limited amount of instance structures which follow a specific pattern, e.g. $A \rightarrow B \rightarrow * \rightarrow$ D would select all instances which used the task A followed by task B followed by any other set of tasks followed by task D.

2.) **Detecting Pharmacological Interactions:** Another application example would be a medical system to support doctors in avoiding dangerous pharmacological interaction. For instance, if a patient has already been prescribed Ciclosporin (an immunosuppressant drug usually used after organ transplants) and now gets a prescription from a different doctor for a barbiturate (drugs that act as central nervous system depressants). Taking both medicins at the same time may have dangerous interactions. To be more precise, a barbiturate negatively influences the effective level of Ciclosporin which may lead to organ repulsion. A sequential analysis would allow doctors to avoid prescribing such combinations of drugs.

3.) **Ticketing systems** for light rail traffic, skiing resorts or multi-storey car parks would be another application example. Here, a user could want to analyze different sets of customers which for instance took a specific route $A \rightarrow * \rightarrow X$, which means that they entered the subway at station A, changed trains at any station, and left the subway at station X.

4.) **Sensor data warehouses** would also be an interesting application area for a sequential data warehousing approach. Consider a data warehouse that stores information which stems from dozens sensors mounted at a power turbine. Analyzing sequences in this data warehouse could provide very useful information, e.g. to reduce down-times. For instance, we could want to analyze the allocation of heat of certain parts of the turbine within 30 seconds after a specific sensor reported a defined temperature.

7 Related Work

While the support of sequential data in traditional database management systems in general and specifically on time-sequences isn't a new topic (see [11], [12], [10], [2]), the term of Sequence OLAP or S-OLAP has been coined recently in [9]. In [9] the authors present an approach where a user defines a query based on pattern templates to analyze sequence data. A pattern template consists of a sequence of symbols where each symbol corresponds to a domain of values. In contrast to a pattern template, e.g. (A, B, A) a pattern is an instantiation of cell values corresponding to a pattern template. A prototypical implementation of such an S-OLAP system has been presented in [3].

The approach presented in [9] has been extended by the same research group in [4]. In [4] the authors focus on the efficient evaluation of ranking pattern based aggregate queries. As in [9] the number of dimensions of the defined cube is equal to the number of distinct values of the selected attribute in the source table.

In order to avoid an overwhelming amount of data to be presented to the user, [4] introduces support for top-k queries.

Another interesting approach has been presented in [8]. The authors combine two existing technologies, namely OLAP (Online Analytical Processing) and CEP (Complex Event Processing) to analyze real-time event data streams. They introduce patterns and pattern hierarchies. If a pattern A contains a subset of event types compared to a pattern B, then A is at a coarser level then B in the resulting pattern hierarchy. Based on these hierarchical relationships, the authors present different strategies how to exploit these hierarchies for query optimization.

The approach presented in [1] discusses a model to analyze time-point-based sequential data. The authors introduce a formal model and define several operators to create and analyze sequences. Furthermore, it formaly defines and discusses the notion of facts, measures and dimensions in the context of sequential OLAP.

Our approach differs from the approaches discussed in this section as follows: our approach is not a redefinition of the well know OLAP approach and architecture as for instance presented in [6], but an extension. To the best of our knowledge, it is the first sequential OLAP approach that smoothly integrates into existing OLAP systems.

8 Conclusion

Traditional data warehouse and OLAP approaches still fail when it comes to efficiently analyze sequential data, i.e. data with a logical or temporal ordering [9]. For instance, a query like "what are the follow-up costs of patients diagnosed cerebral infarction within 12 months after the diagnose" cannot be answered without a relative time axis defined in the data warehouse for the event defined in the query (here: diagnose cerebral infarction). A naive approach to solve this problem would be to create a relative time axis in advance for all combinations of events. However, such a naive approach will fail as the number of possible combinations will quickly blast the capacity of the cube.

In this paper we presented a novel and sophisticated approach that enables the user to analyze sequential data in a standard OLAP environment. The user may state simple queries that require only an atomic event or complex queries with a defined sequence of events. The result of our approach is itself a standard OLAP cube, extended with a new dimension representing the relative time axis. Thus, it is easy to implement our approach into an existing OLAP solution. Furthermore, the user may use her or his OLAP solution to analyze the resulting data.

We implemented this approach as a proof of concept. Basically, this implementation enables the user to define a sequence of events and automatically apply the defined functions $solap()$ and $rta()$ to a given data warehouse.

Future work will focus on wildcard support in sequence definitions. A wildcard may be a question mark "?", represeting any single event, an asterisk "$*$", representing any sequence of events or a plus "$+$", representing any sequence of events which consists of at least one event.

References

1. Bębel, B., Morzy, M., Morzy, T., Królikowski, Z., Wrembel, R.: Olap-like analysis of time point-based sequential data. In: Castano, S., Vassiliadis, P., Lakshmanan, L.V., Lee, M.L. (eds.) ER 2012 Workshops 2012. LNCS, vol. 7518, pp. 153–161. Springer, Heidelberg (2012)
2. Chandra, R., Segev, A.: Managing Temporal Financial Data in an Extensible Database. In: VLDB (1992)
3. Chui, C., Kao, B., Lo, E., Cheung, D.: S-OLAP: an OLAP System for Analyzing Sequence Data. In: SIGMOD (June 2010)
4. Chui, C., Lo, E., Kao, B., Ho, W.: Supporting Ranking Pattern-Based Aggregate Queries in Sequence Data Cubes. In: CIKM (2009)
5. Eder, J., Olivotto, G.E., Gruber, W.: A Data Warehouse for Workflow Logs. In: Han, Y., Tai, S., Wikarski, D. (eds.) EDCIS 2002. LNCS, vol. 2480, pp. 1–15. Springer, Heidelberg (2002)
6. Kimball, R.: The Data Warehouse Toolkit, 2nd edn. John Wiley & Sons (1996)
7. Koncilia, C.: The COMET Temporal Data Warehouse (PhD). In: UMI (2002)
8. Liu, M., Rundensteiner, E., Greenfield, K., Gupta, C., Wang, S., Ari, I., Mehta, A.: E-cube: Multi-dimensional event sequences processing using concept and pattern hierarchies. In: ICDE (2010)
9. Lo, E., Kao, B., Ho, W., Lee, S., Chui, C., Cheung, D.: OLAP on Sequence Data. In: SIGMOD (June 2008)
10. Segev, A., Shoshani, A.: Logical Modeling of Temporal Data. In: SIGMOD (1987)
11. Seshadri, P., Livny, M., Ramakrishnan, R.: Sequence query processing. In: SIGMOD (1994)
12. Seshadri, P., Livny, M., Ramakrishnan, R.: The Design and Implementation of a Sequence Database System. In: VLDB (1996)
13. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow patterns. In: Distributed and Parallel Databases (2003)