# A Relational Data Warehouse for Multidimensional Process Mining

Thomas Vogelgesang and H.-Jürgen Appelrath

Department of Computer Science
University of Oldenburg, Germany
thomas.vogelgesang@uni-oldenburg.de

**Abstract.** Multidimensional process mining adopts the concept of data cubes to split event data into a set of homogenous sublogs according to case and event attributes. For each sublog, a separated process model is discovered and compared to other models to identify group-specific differences for the process. Even though it is not time-critical, performance is vital due to the explorative characteristics of the analysis. We propose to adopt well-established approaches from the data warehouse domain based on relational databases to provide acceptable performance. In this paper, we present the underlying relational concepts of PMCube, a data-warehouse-based approach for multidimensional process mining. Based on a relational database schema, we introduce generic query patterns which map OLAP queries to SQL to push the operations (i.e. aggregation and filtering) to the database management system. We evaluate the run-time behavior of our approach by a number of experiments. The results show that our approach provides a significantly better performance than the state-of-the-art for multidimensional process mining and scales up linearly with the number of events.

## 1 Introduction

Process mining is a set of techniques that allows for the automatic analysis of (business) processes. It is based on so-called event logs which consist of events recorded during the execution of the process. Figure 1 illustrates the typical structure of event logs. The events are grouped by their respective process instance (case) and the ordered sequence of events for a case forms the trace. Both, cases and events, may store arbitrary information as attributes.

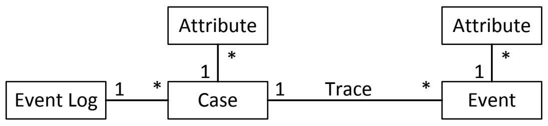

**Fig. 1.** General structure of an event log

Process mining can be categorized in three different kinds: (1) Process discovery extracts a process model from the event log reflecting its behavior, (2) conformance checking compares an event log to a manually created or previously discovered process model to measure the quality of the process model, and (3) process enhancement extends a process model with additional information (e.g., timestamps) to provide additional perspectives on the process.

Besides traditional business processes, process mining can also be applied in the domain of healthcare e.g., to analyze the treatment process in a hospital. However, healthcare processes are typically unstructured due to the individuality of patients. The treatment has to be adjusted to the individual situation of the patient considering age, sex, other diseases, and other features of the patient. Furthermore, the process may also be influenced by institutional features, e.g., the experience of the medical staff. To minimize the influence of such features, it is desirable to group patients with similar features and to analyze the process separately for each group. Otherwise, the heterogeneity of patients would result in a very complex model blurring the influence of particular features.

Traditional process mining techniques only consider the entire event log. Even though the filters can be applied to restrict the event log to a particular subset of cases or events, this requires high effort if multiple groups of patients should be analyzed and compared to each other. Therefore, an approach is required that enables the analyst to partition event logs into groups of cases with homogeneous features in a dynamic and flexible way. Then, an individual process model for each group can be separately mined and compared to other models.

Multidimensional process mining (MPM) achieves this by adopting the concept of data cubes that is well-known in the domain of data warehousing. It considers the attributes of the event log, describing the variable features of the patients, as dimensions forming a multidimensional data space. Each combination of dimension values forms a cell of the cube that contains a subset of the event log (sublog) related to these dimension values. OLAP operators can be used to manipulate the data cube and define a specific views on the data.

MPM is characterized by its explorative approach. The OLAP queries are gradually modified to analyze the processes from multiple views. This way, the analyst can derive and verify new hypothesis. To avoid interruptions of the explorative workflow, it is important to keep waiting times as short as possible. Therefore, performance is vital for MPM, even though it is not a time-critical application. We propose to adopt well-established data warehouse (DWH) technologies based on relational databases, to provide acceptable loading times for the multidimensional event data. In this paper, we show how to link multidimensional event data to relational databases for MPM. As our main contribution, we show how to express the MPM-specific OLAP queries using SQL to push filtering and aggregation of event data into the database management system (DBMS). This way, MPM may benefit from the comprehensive optimization techniques provided by state-of-the-art DBMS.

The paper is organized as follows. First, we discuss related work in Section 2. The general concept of our approach PMCube is briefly introduced in Section 3. While Section 4 presents the logical data model of the underlying data warehouse, Section 5 explains its mapping on a relational database schema. In Section 6, we map high-level OLAP operators to generic patterns expressed in SQL. In Section 7, we evaluate our approach by a number of performance measurements comparing our approach to the state-of-the-art approach for MPM. We conclude our paper in Section 8.

## 2 Related Work

There is a wide range of literature in the data warehousing domain (e.g., [2]) describing the different general approaches for the realization of data cubes. Multidimensional OLAP (MOLAP) approaches rely on a mainly memory-based multidimensional array storage. Relational OLAP (ROLAP) maps the multidimensional data to a relational database schema. A combination of both approaches is known as Hybrid OLAP (HOLAP). In ROLAP approaches, the schema typically consists of a fact table storing the data values. This table is linked to other tables storing the values of the dimension and their hierarchies. In the star schema, each dimension is stored in a single table representing all its dimension levels while the snowflake schema stores each dimension level in its own table.

The process mining manifesto [10] gives an overview of the field of process mining. For a comprehensive introduction to this topic, we refer to van der Aalst [8]. Event Cubes [7] are a first approach for MPM. This approach uses information retrieval techniques to create an index over a traditionally structured event log and derives a data cube from it. Each cell of an event cube contains precomputed dependency measures instead of raw event data. A single process model is generated on-the-fly from these dependency measures where each value of the considered dimensions is mapped to a different path in the model.

Process Cubes [9] are another approach for MPM which uses OLAP operators to partition the event data into sublogs. It combines all dimensions in a common data space so the cells of the cube contain sets of events. Its implementation Process Mining Cube (PMC) [1] can use different algorithms to discover a process model for each cell and provides the visualization of multiple process models in a grid. Both, Event Cubes and PMC, are based on a MOLAP approach. According to Bolt et al. [1], PMC provides a significantly better performance than the previous Process Cubes implementation PROCUBE. However, the reported loading times are still quite long if related to the amount of data. Additionally, the filtering is limited to filtering particular cases and events. Moreover, it does not provide the aggregation of events into high-level events.

Besides, there are several approaches for special DWH for storing process data (e.g. [6,5]). These process warehouses (PWH) aim to analyze the underlying processes to identify problems in process execution. However, they do not store complete event logs, but measures for process performance (e.g. execution times), where events and cases form the dimensions of the cube. The analysis is performed by aggregating the measures along the dimensions. In contrast to MPM, these approaches generally do not support process mining.

## 3 PMCube Concept

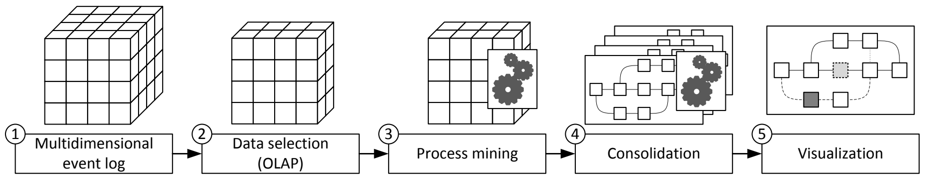

**Fig. 2.** Basic concept of PMCube

Figure 2 illustrates the basic concept of PMCube. Starting point for each analysis is the multidimensional event log (MEL; step ①). It is a specific DWH which stores all the available event data in a cube-like data structure. Section 4 introduces its logical data model while Section 5 presents its relational-based realization.

By using OLAP operators (step ②), it is possible to filter (slice or dice) the MEL or to change its level of aggregation (roll-up or drill-down). This allows for the creation of flexible views on the event data. The query results in a set of cells where each cell contains a sublog. Each sublog is mined separately (step ③) using an arbitrary process discovery algorithm to create an individual process model reflecting the behavior of the sublog. Additionally, it possible to enhance the process model with additional perspectives or to measure its quality using conformance checking techniques.

The OLAP query may result in multiple cells, leading to an unmanageable amount of process models. Therefore, PMCube introduces an optional step of consolidation (step ④), which aims to reduce the complexity of results. Its main idea is to automatically preselect a subset of potentially interesting process models by a heuristic. For example, assuming that big differences are more relevant to the analyst than minor differences between the models, it is possible to cluster similar process models and select a representative for each cluster. Finally, the process models are visualized (step ⑤). As MPM strongly benefits from comparing the different process models, it is not sufficient to visualize each model on its own. Therefore, PMCube provides several visualization techniques, e.g. merging two models into a difference model highlighting the differences between them. The concept of PMCube is presented in [11] in more detail.

## 4 Logical Data Warehouse Model

In contrast to the Process Cubes approach, the MEL maps the structure of event logs to a data cube and organizes cases and events on different levels. Furthermore, the cells of the MEL do not contain sets of events, but a set of cases. The attributes of the cases are considered as dimensions forming the multidimensional data cube. Each combination of dimension values identifies a cell of the cube. According to the values of its attributes, each case is uniquely mapped to a cell. However, some attributes represent highly individual features, e.g., the name of the patient. Mapping them to dimensions results in sparse data cubes and does not add any benefit to the multidimensional analysis. On the contrary, these attributes might give valuable insights if the process model behavior is related to individual cases. Therefore, these non-dimensional attributes are directly attached to the case as so-called simple attributes.

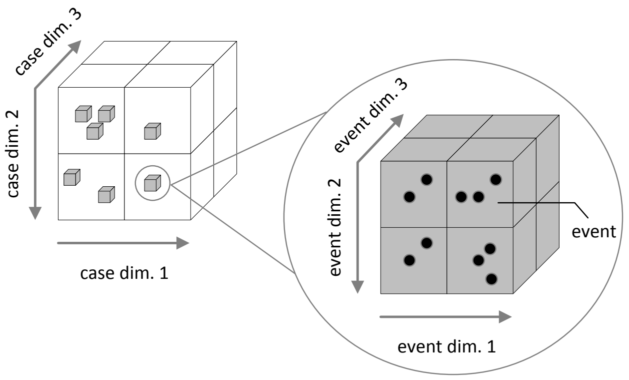

**Fig. 3.** Nested Cubes

Related to their respective case, the events are stored inside the MEL as well. Similar to the case attributes, the event attributes can be interpreted as dimensions, too. To avoid the aggregation of events from different, independent cases, the event attributes form an individual multidimensional data space for each case which is contained in the cell of the case. Figure 3 shows the relationship of these nested cubes. Each cell of the data cubes on the event level consists of a set of events identified by a combination of event dimension values. Similar to cases, non-dimensional event attributes are directly attached to the event as simple attributes. All dimensions, both on the case and event level, may have an arbitrary number of hierarchically structured dimension levels.

The OLAP queries like slice, dice, roll-up and drill-down are defined on a set of base operators like filtering (selection) and aggregation. Due to different semantics, the definition of the operators might vary between case and event level. Figure 4 illustrates this using the example of the aggregation operator. Aggregating cells on the case level creates the union of all the cells cases. For example, aggregating the cube on the left-hand of Figure 4 along the dimensions *sex* and *age* results in a single cell containing all cases for both women and men of all age for a specific year. On the contrary, aggregating cells on the event level merges all events into a single, more abstract event. This is demonstrated on the right-hand of Figure 4, showing the aggregation of events along the dimensions *activity* and *resource*. Previously, various events are spread across different cells,

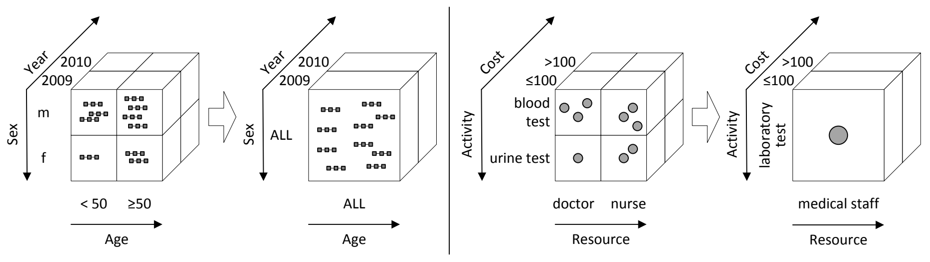

**Fig. 4.** Aggregation of cases (left) and events (right)

each representing different kinds of activities performed by either doctors or nurses. The aggregation abstracts from the original events and replaces them by a single merged event. This style of aggregation can be useful if the analyst is only interested if a laboratory test was performed or not, regardless of which kind of test or how many tests were performed. Reducing the number of events may simplify the mined process model by reducing its number of nodes.

The MEL can be filtered by selection operators on both the case and the event level. On the event level, the selection predicate contains only event attributes (dimensions as well as simple attributes). This allows, for example, to remove all events representing non-medical activities to focus on the medical treatment process. On the case level, the MEL can be filtered by both case and event attributes. However, a quantifier ($\exists$ or $\forall$) must be specified for each event attribute of the selection predicate in order to specify whether the condition must hold for at least one event or for all events of a case. Alternatively, an aggregation function (*min*, *max*, *avg*, *sum*, or *count*) can be specified, e.g. to select only cases exceeding a maximum cost limit.

The MEL typically contains all available case and event attributes. However, most process mining techniques (i.e. process discovery algorithms) only need a small subset of attributes. To reduce the amount of data loaded from the MEL, the projection operator can be used to remove unneeded attributes. In contrast to Process Cubes, the data model of our approach is a little more restrictive, e.g. it is not possible to change the case id during the analysis. However, it allows for a wider range of operations (e.g., selecting full cases based on event attributes) and a clear mapping to the relational data model which is discussed in the following section.

## 5 Relational Data Warehouse Model

In contrast to traditional data cubes, the cells of the MEL do not contain single values but complex data. As available data warehousing tools are not capable of handling such data, MPM requires specific solutions storing event log data. The cells of the MEL consist of an arbitrary number of cases and events, which contradicts the MOLAP approach, where each cell typically represents a data point of fixed size. In contrast, ROLAP approaches allow for a more flexible modeling of complex data. Additionally, a ROLAP-based approach for MPM can benefit from various optimization techniques implemented in state-of-the-art DBMS. Therefore, we choose a ROLAP approach to realize the MEL.

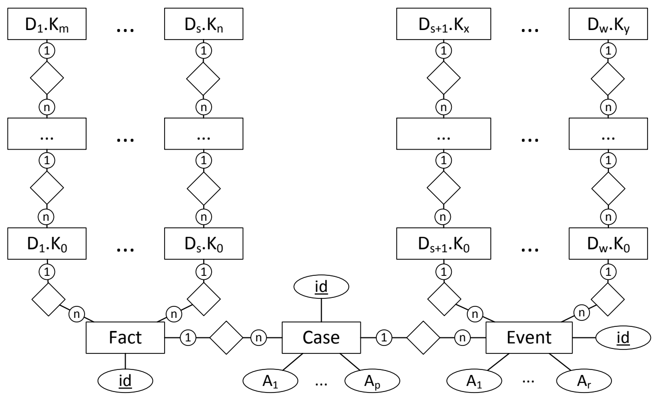

**Fig. 5.** Generic database schema of the MEL

Even though star schemes usually provide a better performance, we extend the traditional snowflake schema to avoid redundancy which may lead to data anomalies. Figure 5 shows the generic database schema as an entity-relationship model. Similar to the traditional snowflake schema, there is a *fact* table for storing the cells of the data cube. Each cell is identified by a unique combination of foreign keys referencing the cells dimension values. These values are stored in normalized dimension tables (e.g., $D_1.K_0$ to $D_1.K_m$ for a dimension $D_1$) to avoid redundancy. In contrast to the traditional snowflake schema, the fact table does not directly store the cells value, but a unique *id*. The data content of the cells, namely the cases, is normalized and stored in the *case* table, which also stores the simple attributes ($A_1$ to $A_p$) of the cases. The corresponding cell of a case is referenced by the *fact id*. The events are normalized in an analogous manner and stored in the *event* table, which also holds the simple attributes of the event. Events can also have dimension attributes, which are stored in dimension tables similar to the case dimensions. However, the event table directly references the dimension tables, as dimension values might differ for events of the same case.

Figure 6 illustrates how the event data is loaded from the MEL and processed in PMCube. Starting point is an OLAP query which is defined by a user, e.g., through a graphical user interface (cf. ①). By this OLAP query, the user describes a logical view on the data cube (cf. ②). After that, the OLAP query is translated into a set of SQL queries (cf. ③ and ④). Each SQL query represents a cell defined by the OLAP query and expresses the appropriate filtering and aggregation operations. Section 6 presents the query translation in more detail. The SQL queries are send to the MEL consisting of a relational database (cf.

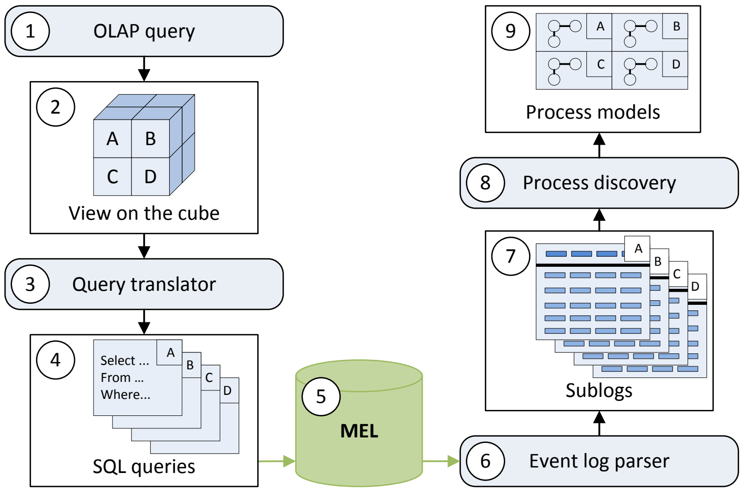

**Fig. 6.** Mapping an OLAP query to a set of SQL queries

⑤). The result of each query is a set of tuples, each tuple representing an event with all its (case and event) attributes. Immediately after the query result is sent back, the tuple set is parsed (cf. ⑥) and translated into a sublog (cf. ⑦) with the usual event log structure (cf. Figure 1). Then the sublogs are mined using an arbitrary process discovery algorithm (cf. ⑧). To improve the performance, the sublogs are parsed and mined asynchronously. This means, that the data is processed immediately after it has been loaded from the MEL. Finally, the process discovery results in a set of process models (cf. ⑨), one model for each cell.

Typically, the events in the traces of an event log are temporally ordered. This is mandatory to identify relationships between particular events. File-based event log formats like XES [3] usually imply this order by the position in the documents structure. However, relational databases store unordered multisets of tuples. To be able to restore the event order, PMCube requires the definition of an order-preserving attribute. By default, we use the event timestamp for this. However, it might be possible that the timestamp is missing or not precise enough to preserve the event order. Therefore, it is also possible to use other attributes e.g., an explicit event index or the event id if this reflects the order of events.

## 6 Generic Query Patterns

PMCube aims to benefit from the various optimization techniques of state-of-the-art relational DBMS providing high performance and scalability. Therefore, PM-Cube expresses the filtering and aggregation operators within the SQL queries to push their processing to the database. PMCube uses a generic query pattern to map the cells defined by the OLAP query to correspondent SQL queries. Listing 1.1 shows the general structure of the generic SQL pattern. To improve the understandability, we use placeholders (<. . . >) for particular parts of the pattern which will be explained in more detail.

```
1 SELECT <event log attributes>
2 FROM Fact
3      JOIN Case  ON Case.fact_id = Fact.id
4      JOIN Event ON Case.id = Event.case_id
5      <dimension joins>
6 WHERE <filter conditions>
7 ORDER BY Event.case_id, <sequence-preserving attribute>
```

**Listing 1.1.** Generic query pattern in SQL-like pseudo code

The placeholder <*event log attributes*> (line 1) is replaced by a list of all database attributes that should be loaded from the database. These database attributes can comprise values of dimension attributes and non-dimensional attributes, both on the case and event level. Representing the MEL's projection operator, it is possible to omit unneeded attributes by specifying a subset of attributes. This reduces the size of the data to be loaded from the database, leading to faster responses, especially if the data is transferred via a low bandwidth network connection. However, the case id, the sequence-preserving attribute, and the classifier (representing the name of the nodes in the process model) are mandatory and must be contained in the attribute list.

As the event data is spread across multiple database tables, it is necessary to join the tables to reconstruct the relationships between them. Therefore, the central tables (*fact*, *case*, and *event*) are joined (lines 2-4). Additionally, the fact table and the event table need to be joined with the dimension tables, to link the events with their respective dimension level values. The placeholder <*dimension joins*> (line 5) subsumes these join statements. Because join operations are very costly, we limit them to the tables that are required to filter the data or to retrieve the attributes specified in <*event log attributes*> (line 1). All other tables are omitted from <*dimension joins*> during query translation.

The placeholder <*filter conditions*> (line 6) subsumes all filtering operations, both on the case and the event level, as a conjunction of boolean expressions. Because each SQL query represents a particular cell, the dimensions forming the cube must be limited to their respective dimension values for this cell. For example, if a cell should represent all patients of the year 2015, <*filter conditions*> must contain an expression like `DIM_TIME_YEAR.VALUE = 2015` (assuming that `DIM_TIME_YEAR` is the name of the table representing the time dimension at the year level and that `VALUE` is the name of an attribute of this table storing the values of the year). Finally, the tuples of the result table are sorted by the case id and the sequence-preserving attribute (line 7). This is done to restore the sequence of events for each case.

To filter cases by the attributes of their events, the <*filter conditions*> in Listing 1.1 (line 6) need to be extended by a subquery. The subquery selects the case ids of all cases meeting a particular condition. Due to the different kinds of case selection over event attributes (∃, ∀, aggregation), there are differences in the patterns for the subqueries as well. Listing 1.2 shows the subquery for the selection of cases with at least one event per case matching a condition. It simply selects all case ids of an event meeting the boolean condition given in line 3 (<*condition*>). Duplicates are removed using the `UNIQUE` key word, because more than one event of a case might match the condition.

```
1  ... AND case_id IN (
2              SELECT UNIQUE case_id FROM Event
3              WHERE <condition>
4      ) ...
```

**Listing 1.2.** Subquery for selecting cases with at least one event matching a condition

If the condition must hold for each event of a case, the subquery shown in Listing 1.3 is used. Because SQL does not support such a selection, we use double negation. First, we select the ids of all cases that violate the condition expressed in <*condition*> (line 3). At this point, we also have to check all variables <*v1*> to <*vn*> used in <*condition*> for NULL values (lines 4-6). This is required because undefined attribute values are a violation of the condition as well which is however not covered by the condition in line 3. After we have selected the ids of all cases violating the condition, we only select that cases not contained in this subset (line 1).

```
1  ... AND case_id NOT IN (
2              SELECT UNIQUE case_id FROM Event
3              WHERE NOT <condition>
4                        OR <v1> IS NULL
5                        OR ...
6                        OR <vn> IS NULL
7      ) ...
```

**Listing 1.3.** Subquery for selecting cases where each event of a case matches a condition

Furthermore, PMCube allows for the selection of cases by aggregated event attributes. Assuming each event has an attribute representing the individual cost for the execution of its activity, it is possible to select all cases that e.g., have at least an average cost per event of 100$. This allows the analyst to define a very specific filtering of cases. Listing 1.4 shows the subquery to express this kind of filtering. The subquery groups all events by the id of their cases (line 3). After that, the condition is evaluated for each case. The placeholder <*condition*> (line 4) consists of a boolean expression which specify an aggregation over the grouped events. It is possible to use arbitrary SQL aggregation functions like SUM, AVG, MIN, or MAX for any event attribute.

Finally, it is also possible to realize the aggregation of events (cf. Section 4) within the SQL query. For this operation, we extend the generic query pattern

```
... AND case_id IN (
            SELECT UNIQUE case_id FROM Event
            GROUP BY case_id
            HAVING <condition>
    ) ...
```

**Listing 1.4.** Subquery for selecting cases using aggregations over event attributes

from Listing 1.1 at several points. First, we insert a `GROUP BY` *<attributes>* statement between lines 6 and 7 to group the attributes that should be merged into a single high-level attribute. To avoid mixing events from different cases, the attribute list <*attributes*> starts with the case id. Additionally, the list comprises all dimension attributes at their respective dimension level that should be targeted by the aggregation. Note that omitting a particular dimension from <*attributes*> rolls up the data cube to the artificial root node ALL which describes the highest level of abstraction comprising all values of a dimension. E.g., inserting the statement `GROUP BY Event.case_id, Dim_Activity.id` aggregates all events of a case that represent the same activity to a new high-level activity.

The aggregated events have the same structure as the original events. Therefore, the event attributes of the original attributes have to be aggregated into a single value for each aggregated event. The aggregation of the dimension attributes is given by the structure of the dimension hierarchies. For each non-dimensional attribute, we individually select an SQL aggregation function depending on the semantics of the attribute. E.g., for the attribute *cost of activity* it makes sense to sum up the individual costs of the original events. This way, the new value will reflect the total cost of all events that are comprised by the aggregated event. However, depending on the analysis question, also other aggregation functions (e.g., average) might be meaningful, so it is a partially subjective choice.

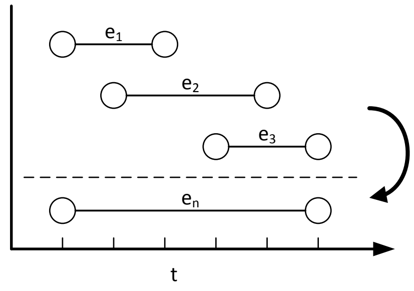

**Fig. 7.** Aggregating start and end timestamps of events

Figure 7 illustrates the merging of the start and end timestamps of events as another example. Using the MIN function for the start timestamp and the MAX function for the end timestamp ensures, that the aggregated event $e_n$ covers the same period of time like the single events $e_1$, $e_2$, and $e_3$. However, there might be some event attributes that cannot be aggregated in a meaningful way. To ensure that these attributes do not impede the aggregation, we propose to use the MIN function to aggregate the attributes even though these attributes will probably not contribute to the interpretation of results anymore. Additionally, all case attributes in the attribute list <*event log attributes*> (cf. Listing 1.1, line 1) have to be surrounded by aggregation function. This is due to the fact that SQL only allows for aggregations and grouping attributes after the SELECT statement if a grouping is used. We propose to use the MIN function for them, because all case attributes have the same value for each event of a case and the MIN function will preserve this value.

## 7 Evaluation

We implemented our approach in a prototype called PMCube Explorer [12]. Due to a generic, plug-in-based interface, arbitrary DSMS can be used to store the data. We conducted a case study and a number of experiments to evaluate our approach. We reported the results of the case study in [11]. In this section, we focus on the experiments measuring the run-time performance of our approach.

For our experiments, we used the data set of the case study. It consists of 16,280 cases and 388,395 events and describes a process in a large German hospital of maximum care. For a more detailed description of the data, we refer to Vogelgesang et al. [11]. We created multiple subsets of that data, each consisting of a particular number of events. To evaluate PMCube, we also performed similar tests with the PMC tool[1] as the state-of-the-art implementation of the Process Cubes approach. Event Cubes were not considered in the experiments because that approach is very different (e.g, no creation of sublogs) which makes it incomparable to PMCube and Process Cubes. All experiments were conducted on a laptop with Intel Core i5-2520M 2,5 GHz CPU, 16 GB DDR3 RAM, and ATA SSD running on 64-bit Windows 7. For the MEL, we used a local installation of Microsoft SQL Server 2012. We used the following OLAP queries derived from our case study.

- Q1: We used the dimensions *medical department* and *urgency* to create the cube and removed all cases with the value *unknown* for the *urgency* dimension.
- Q2: The dimensions *medical department* and *reason for discharge* form the cube. No filtering was used.
- Q3: The dimensiones *age* (in 10-year-classes) and *gender* form the cube. All cases with values *unknown* and *older than 100 years* were removed.
- Q4: We used the dimensions *urgency* and *type of admittance* (in-patient or not) to create the cube, filtering the value *unknown* for both dimensions.

To ensure comparability of both tools, we measured the total run-time for the processing of the OLAP query, discovering the model using Inductive Miner [4], and visualizing the process models of all cells in a matrix using process trees. Because some preliminary test runs with small data sets indicated unacceptable run-times of several hours for PMC with bigger data sets, we only used the minimum set of dimensions for PMC to improve its performance. This means, that we omitted all dimensions which were not used to express the OLAP query. However, for our approach, PMCube we used all available dimensions even though

[1] `http://www.win.tue.nl/~abolt/userfiles/downloads/PMC/PMC.zip`, downloaded on June, 16th 2015

| Query | Events (in thousand) | 50 | 100 | 150 | 200 | 250 | 300 | 350 | 388.395 |
|---|---|---|---|---|---|---|---|---|---|
| Q1 | PMC (min) | 28.7 | 98.6 | 212.9 | 356.2 | 560.6 | 792.8 | 1013.0 | 1263.3 |
| | PMCube Explorer (seq) | 7.1 | 11.0 | 15.5 | 20.5 | 25.6 | 30.4 | 35.5 | 38.8 |
| | PMCube Explorer (async) | 5.0 | 7.3 | 10.2 | 13.5 | 17.1 | 20.3 | 24.5 | 26.4 |
| Q2 | PMC (min) | 31.7 | 108.9 | 222.3 | 387.1 | 561.2 | 800.2 | 1087.2 | 1319.9 |
| | PMCube Explorer (seq) | 8.7 | 13.3 | 19.6 | 26.1 | 30.5 | 36.1 | 40.7 | 45.0 |
| | PMCube Explorer (async) | 7.3 | 10.0 | 14.6 | 18.5 | 21.1 | 24.6 | 27.3 | 29.8 |
| Q3 | PMC (min) | 31.0 | 101.8 | 214.1 | 363.1 | 549.8 | 761.4 | 1043.4 | 1302.7 |
| | PMCube Explorer (seq) | 8.3 | 12.1 | 17.0 | 21.8 | 26.9 | 31.7 | 36.8 | 40.1 |
| | PMCube Explorer (async) | 6.5 | 8.9 | 11.4 | 14.1 | 17.3 | 20.1 | 22.4 | 25.3 |
| Q4 | PMC (min) | 28.0 | 96.9 | 203.5 | 350.1 | 534.2 | 755.5 | 1021.0 | 1269.3 |
| | PMCube Explorer (seq) | 4.7 | 8.7 | 13.5 | 18.9 | 24.7 | 30.1 | 35.9 | 41.6 |
| | PMCube Explorer (async) | 3.7 | 6.9 | 10.7 | 15.4 | 20.2 | 25.4 | 30.0 | 35.0 |

**Table 1.** Average run-times in seconds

this might overrate the performance of PMC. Table 1 shows the average run-time in seconds of ten runs for both tools and each combination of query and data set. To evaluate the performance improvement of the asynchronous process discovery, we also performed the experiments with an alternative configuration of PMCube Explorer with a sequential processing of cells. Note that the last column of Table 1 shows the measured values for the full data set.

The values in Table 1 show that the measured run-times of PMCube Explorer, both for asynchronous as well as sequential processing of cells, are significantly shorter than the run-times of PMC. E.g., PMCube Explorer needs between 25 and 45 seconds to process the queries for the complete data set while PMC requires more than 21 minutes for it.

Figure 8 shows the run-time of query Q1 over the number of events. It reveals a polynomial incline for the PMC tool and a linear incline for the PMCube Explorer with asynchronous process discovery. Figure 9 compares the run-time of both configurations of PMCube Explorer for query Q1. It confirms that the run-time increases linearly by the number of events. Additionally, it clearly shows the performance benefit of the asynchronous processing of events which increases by the number of events as well. The measurements for the queries Q2, Q3, and Q4 show a similar run-time behavior indicating the performance advantage over the state-of-the-art. However, we omit the corresponding charts due to space limitations.

## 8 Conclusions

Multidimensional process mining adopts the concept of data cubes to the field of process mining. Even though it is not time-critical, performance is a vital aspect due to its explorative characteristics. In this paper, we presented the realization of our approach PMCube using a relational DBMS. The logical data model is mapped to a generic relational database schema. We use generic query

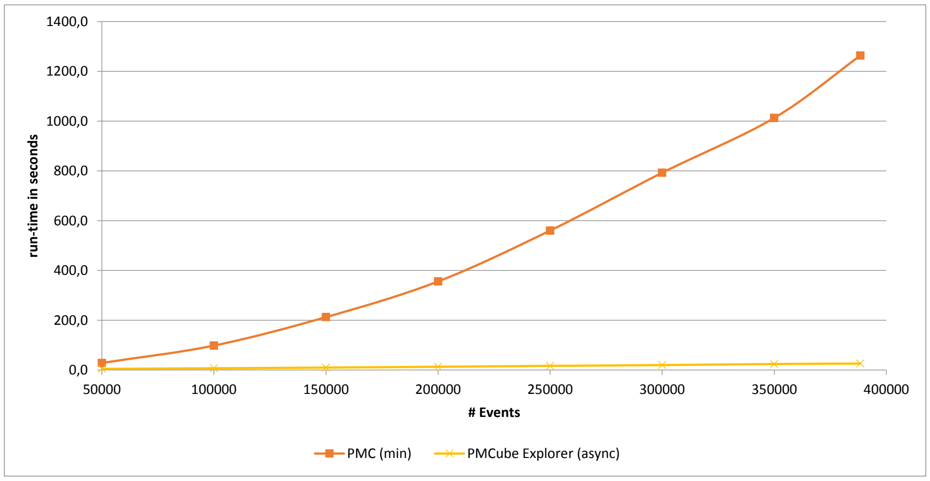

**Fig. 8.** Comparing average run-time of PMCube Explorer (async) with PMC (min) for query Q1

patterns to express the OLAP queries by a separated SQL query for each cell. The experiments reported in this paper show, that PMCube provides a significantly better performance than PMC, the state-of-the-art implementation of the Process Cubes approach. Additionally, the performance of our approach seems to scale linearly by the number of events, promising acceptable processing times with bigger amounts of data. Nevertheless, further improvements of performance might be possible, e.g., by denormalizing the relational schema (similar to a star schema), which should be evaluated by future research.

## References

1. Bolt, A., van der Aalst, W.M.: Multidimensional Process Mining Using Process Cubes. In: Gaaloul, K., Schmidt, R., Nurcan, S., Guerreiro, S., Ma, Q. (eds.) Enterprise, Business-Process and Information Systems Modeling, Lecture Notes in Business Information Processing, vol. 214, pp. 102–116. Springer International Publishing (2015)
2. Golfarelli, M., Rizzi, S.: Data Warehouse Design: Modern Principles and Methodologies. McGraw-Hill, Inc., New York, NY, USA, 1 edn. (2009)
3. Günther, C.W.: XES Standard Definition (Mar 2014), `http://www.xes-standard.org/xesstandarddefinition`
4. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering Block-Structured Process Models from Event Logs - A Constructive Approach. In: Colom, J.M., Desel, J. (eds.) Petri Nets. Lecture Notes in Computer Science, vol. 7927, pp. 311–329. Springer (2013)
5. Neumuth, T., Mansmann, S., Scholl, M.H., Burgert, O.: Data Warehousing Technology for Surgical Workflow Analysis. In: Proceedings of the 2008 21st IEEE International Symposium on Computer-Based Medical Systems. pp. 230–235. CBMS '08, IEEE Computer Society, Washington, DC, USA (2008)

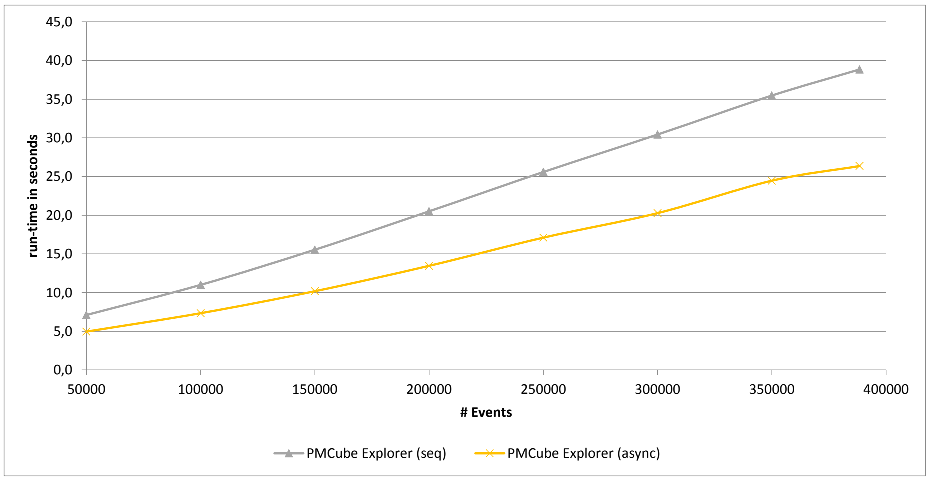

**Fig. 9.** Comparing average run-time of PMCube Explorer (async) with PMCube Explorer (seq) for query Q1

6. Niedrite, L., Solodovnikova, D., Treimanis, M., Niedritis, A.: Goal-driven design of a data warehouse-based business process analysis system. In: Proceedings of the 6th Conference on 6th WSEAS Int. Conf. on Artificial Intelligence, Knowledge Engineering and Data Bases - Volume 6. pp. 243–249. AIKED'07, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA (2007)
7. Ribeiro, J.T.S., Weijters, A.J.M.M.: Event cube: another perspective on business processes. In: Proceedings of the 2011th Confederated international conference on On the move to meaningful internet systems - Volume Part I (OTM'11). pp. 274–283. Springer-Verlag, Berlin, Heidelberg (2011)
8. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer, Berlin (2011)
9. van der Aalst, W.M.P.: Process Cubes: Slicing, Dicing, Rolling Up and Drilling Down Event Data for Process Mining. In: Song, M., Wynn, M., Liu, J. (eds.) Asia Pacific Business Process Management, Lecture Notes in Business Information Processing, vol. 159, pp. 1–22. Springer International Publishing (2013)
10. van der Aalst, W.M., et al.: Process Mining Manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) Business Process Management Workshops (1). Lecture Notes in Business Information Processing, vol. 99, pp. 169–194. Springer (2011)
11. Vogelgesang, T., Appelrath, H.J.: PMCube: A Data-Warehouse-based Approach for Multidimensional Process Mining. In: Business Process Management Workshops - BPM 2015 International Workshops (Pre-Proceedings) (2015)
12. Vogelgesang, T., Appelrath, H.: Multidimensional Process Mining with PMCube Explorer. In: Daniel, F., Zugal, S. (eds.) Proceedings of the BPM Demo Session 2015 Co-located with the 13th International Conference on Business Process Management (BPM 2015), Innsbruck, Austria, September 2, 2015. CEUR Workshop Proceedings, vol. 1418, pp. S. 90–94. CEUR-WS.org (2015), `http://ceur-ws.org/Vol-1418/paper19.pdf`