A Relational Data Warehouse for Multidimensional Process Mining

Thomas Vogelgesang, H.-Jürgen Appelrath

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. For an effective explorative process analysis, 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 onto 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 comprises 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.

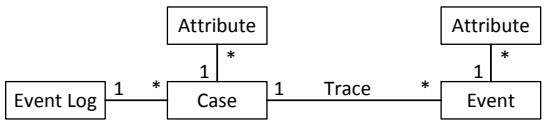


Fig. 1: General structure of an event log

According to van der Aalst [13], there are three different kinds of process mining: (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., time-stamps) 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 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 (Online Analytical Processing) operators [3] can be used to manipulate the data cube and define specific views on the data. For instance, roll-up and drill-down operators can be applied to change the granularity of the cube's dimensions while slice and dice operators can be used to filter 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 satisfying 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 and introduce a running example in Section 3. The general concept of our approach PMCube is briefly introduced in Section 4. While Section 5 presents the logical data model of the underlying data warehouse, Section 6 explains its mapping onto a relational database schema. In Section 7, we map high-level OLAP operators onto generic patterns expressed in SQL. In Section 8, 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 9.

2 Related Work

There is a wide range of literature in the data warehousing domain (e.g., [3]) describing the different general approaches for the implementation of data cubes. Multidimensional OLAP (MOLAP) approaches rely on a mainly memory-based multidimensional array storage. Relational OLAP (ROLAP) maps the multi-dimensional data onto 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 [15] gives an overview of the field of process mining. For a comprehensive introduction to this topic, we refer to van der Aalst [13]. Event Cubes [11] 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 onto a different path in the model.

Process Cubes [14] 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. [9,8]). 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.

Relational XES [16] is a data format for storing event logs which is based on the XES [4] event log structure. In contrast to it, Relational XES uses a relational database in the background to store the data in a relational structure. In comparison to the usual XML-based XES format, Relational XES reduces the overhead for storing big event logs. However, it does not provide a multidimensional structure to store the event logs in a data cube to directly support MPM.

Schönig et al. present an approach for declarative process mining [12] that assumes the data to be stored in a database using the Relational XES schema. In order to use the optimization concepts provided by the database, they define query patterns in SQL to extract the constraints of the declarative process model directly from the database. In contrast to our approach, they perform process discovery directly in the database system which is however limited to declarative process models. Furthermore, they do not either consider multidimensional aspects of the data nor compare different variants of the same process.

An approach for data-aware declarative process mining is described in [7]. It extends the logical constraints of the declarative process model notation by additional conditions defined over the data attributes. Roughly, this approach is comparable to process discovery with an integrated analysis of decision points. Even though it considers arbitrary data attributes of the event logs, it does not provide a multidimensional view on the processes. In contrast to the most approaches for MPM, it generates a single process model instead of a set of variant-specific models.

The approach presented in [10] aims to automatically analyze the structure of relational database items like tables, primary keys, and foreign key relationships to identify so-called business objects or artifacts. Based on the identified artifacts, it automatically generates SQL queries to extract event logs from the database that reflect different processes and are suitable for process mining. However, this approach does not consider MPM.

The Slice-Mine-and-Dice (SMD) approach [2] aims to reduce the complexity of process models in two ways. On the one hand, it clusters similar traces of the event log to split the model into a set of behavioral variants. On the other hand, it reduces the model complexity by extracting and merging of shared subprocesses. Even though the name of the approach indicates a relationship to MPM, it neither considers attribute values nor it uses data cubes or OLAP operations.

3 Running Example

To provide a descriptive explanation of the developed concepts, we introduce a running example in this section. We assume a simple and fictive healthcare process which comprises the diagnoses and therapy for a patient. Figure 2 shows this process as a BPMN model.

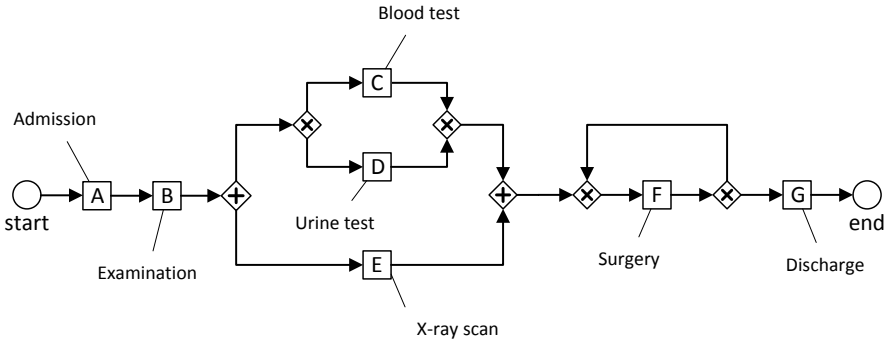


Fig. 2: Simple and fictive healthcare process of the running example

The process starts with the admission of the patient to the hospital (A). After the examination of the patient (B), the process splits into two parallel paths. The first path consists of a lab test, which can be either a blood test (C) or an urine test (D). The second path consists of a X-ray scan (E). After both parallel paths are completed, the treatment proceeds with a surgery (F) which can be repeated multiple times, e.g., in case of complications. Finally, the process ends after the patient's discharge from hospital (G).

For the running example, we define each hospital stay as an independent process instance. Therefore, a particular patient can be represented by several cases. We assume that each case is recorded in an event log with additional information. Age, sex, and insurance status of the patients, year of treatment, and the hospital are saved for the cases. Besides the activity of each event, it is also stored who performed the activity (a doctor or a nurse) and the costs related to the execution of the activity. Furthermore, we assume that each event has a time-stamp which is precise enough to reflect a unique event order.

Table 1 shows an example event log consisting of 20 cases reflecting the fictive example process of Figure 2. For each case, its id and the age, the sex, the insurance status (public or private insurance), the year of treatment, and the hospital location are given. Furthermore, the table shows the trace of each case as the sequence of events using the abbreviations of activity names introduced in Figure 2, e.g, ABCEFFG for the first case. In total, the event log contains four different traces representing different process behavior (variations are underlined): AB<u>C</u>E<u>F</u>G, ABE<u>DF</u>G, AB<u>C</u>E<u>FF</u>G, ABE<u>DFF</u>G. For the sake of clarity, we omit the event attributes.

case id	Age	Sex	Insurance	Year	Hospital	Trace
1	34	female	private	2012	Oldenburg	ABCEFFG
2	31	male	private	2013	Vienna	ABCEFFG
3	23	female	public	2014	Oldenburg	ABCEFG
4	56	male	public	2012	Oldenburg	ABEDFG
5	69	female	public	2015	Vienna	ABEDFG
6	56	male	public	2014	Vienna	ABEDFG
7	54	female	private	2012	Vienna	ABEDFFG
8	55	male	public	2015	Oldenburg	ABEDFG
9	36	female	public	2015	Vienna	ABCEFG
10	67	male	public	2014	Oldenburg	ABEDFG
11	56	female	private	2013	Oldenburg	ABEDFFG
12	68	male	private	2014	Berlin	ABEDFFG
13	30	female	public	2014	Vienna	ABCEFG
14	71	male	public	2012	Berlin	ABEDFG
15	65	female	public	2015	Berlin	ABEDFG
16	23	male	private	2012	Vienna	ABCEFFG
17	29	female	public	2011	Vienna	ABCEFG
18	72	male	private	2010	Oldenburg	ABEDFFG
19	38	female	public	2010	Vienna	ABCEFG
20	42	male	public	2011	Berlin	ABCEFG

Table 1: Example Event Log L_1

Even though the process model (Figure 2) reflects the overall process, it does not reveal the influence of particular attributes. Consequently, the specific process behavior of particular groups of patients cannot be seen in the model. Partitioning the event log L_1 into sublogs using MPM enables the analyst to identify differences in the group-specific process models that are related to particular attributes and their values. Tables 2 and 3 show the two sublogs L_2 and L_3 which can be derived from L_1 using MPM by drilling-down the data cube alongside the age dimension. While L_2 comprises all cases of younger patients (age < 50), L_3 only contains the cases of older patients. For the sake of clarity, we only show the age, sex, and insurance status attributes.

Mining both sublogs results in the two variants of the process model which are shown in Figure 3a (for L_2) and in Figure 3b (for L_3). Comparing these models clearly shows that patients are treated differently depending on their age: The activity C (blood test) is only performed for patients younger than 50 years, the activity D (urine test) is exclusively performed for patients of 50 years or older.

Alternatively, drilling down alongside the insurance status dimension instead of the age dimension would reveal that activity F (surgery) is only repeated for patients with private healthcare insurance. For patients with public healthcare insurance, this activity is only executed once. However, partitioning the event log does not always result in such obvious differences. For instance, the sex of the

case id	Age	Sex	Ins.	Trace
1	34	female	private	ABCEFFG
2	31	male	private	ABCEFFG
3	23	female	public	ABCEFG
9	36	female	public	ABCEFG
13	30	female	public	ABCEFG
16	23	male	private	ABCEFFG
17	29	female	public	ABCEFG
19	38	female	public	ABCEFG
20	42	male	public	ABCEFG

Table 2: Sublog L_2 (age < 50)

case id	Age	Sex	Ins.	Trace
4	56	male	public	ABEDFG
5	69	female	public	ABEDFG
6	56	male	public	ABEDFG
7	54	female	private	ABEDFFG
8	55	male	public	ABEDFG
10	67	male	public	ABEDFG
11	56	female	private	ABEDFFG
12	68	male	private	ABEDFFG
14	71	male	public	ABEDFG
15	65	female	public	ABEDFG
18	72	male	private	ABEDFFG

Table 3: Sublog L_3 (age ≥ 50)

patients does not have any influence on the treatment of the example process. Using the sex of the patients for drilling down results in the two sublogs L_4 (female) and L_5 (male) which are shown in Tables 4 and 5, respectively. Both sublogs comprise all four variations of the process. Consequently, mining the sublogs L_4 and L_5 will result in the overall process model shown in Figure 2.

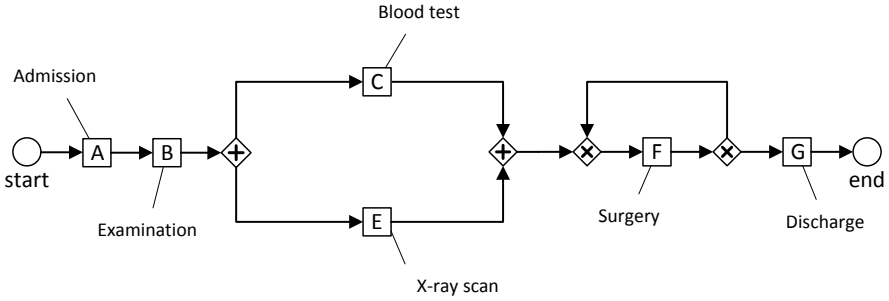
case id	Age	Sex	Ins.	Trace
1	34	female	private	ABCEFFG
3	23	female	public	ABCEFG
5	69	female	public	ABEDFG
7	54	female	private	ABEDFFG
9	36	female	public	ABCEFG
11	56	female	private	ABEDFFG
13	30	female	public	ABCEFG
15	65	female	public	ABEDFG
17	29	female	public	ABCEFG
19	38	female	public	ABCEFG

Table 4: Sublog L_4 (female patients)

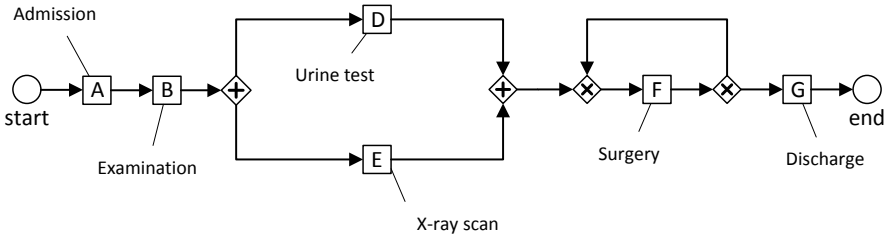
case id	Age	Sex	Ins.	Trace
2	31	male	private	ABCEFFG
4	56	male	public	ABEDFG
6	56	male	public	ABEDFG
8	55	male	public	ABEDFG
10	67	male	public	ABEDFG
12	68	male	private	ABEDFFG
14	71	male	public	ABEDFG
16	23	male	private	ABCEFFG
18	72	male	private	ABEDFFG
20	42	male	public	ABCEFG

Table 5: Sublog L_5 (male patients)

In contrast to the given example, the influence of the dimensions is usually not that clear for real-world data. Even if a dimension has an affect on the process, the sublogs may contain noise due to exceptional behavior. The influence of a dimension may also not be distinctive. This means that the dimension value does not necessarily imply an exclusive choice like always executing activity C (blood test) for younger patients and activity D (urine test) for older patients, as it was shown in the example. Instead, the dimensions often influence the likelihood for



(a) Process model variant for sublog L_2 (age < 50)



(b) Process model variant for sublog L_3 (age ≥ 50)

Fig. 3: Process model variants for sublogs L_2 and L_3

a particular path in the process. Furthermore, process variations may only occur for a specific value combination of multiple dimensions.

4 PMCube Concept

Figure 4 illustrates the basic concept of PMCube. The 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 5 introduces its logical data model while Section 6 presents its relational-based realization.

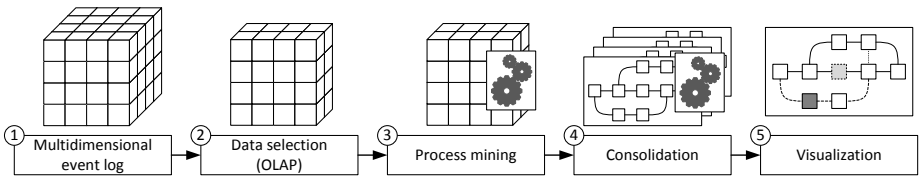


Fig. 4: Basic concept of PMCube

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 is 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. Alternatively, the process models can be selected by simply filtering them by their properties (e.g., occurrence of particular nodes). 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 [18] in more detail.

5 Logical Data Warehouse Model

In contrast to the Process Cubes approach, the MEL maps the structure of event logs onto 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 onto a cell. However, some attributes represent highly individual features, e.g., the name of the patient. Mapping them onto 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.

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 5 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

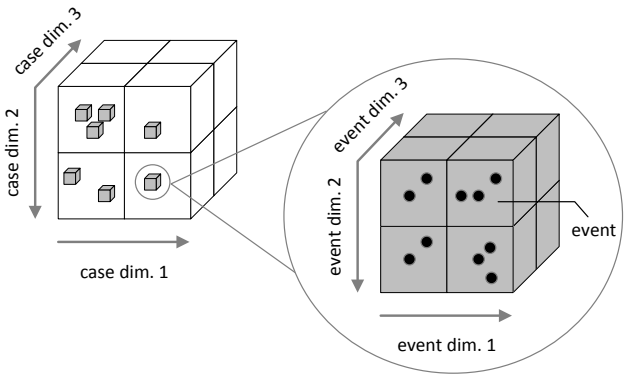


Fig. 5: Nested Cubes

simple attributes. All dimensions, both on the case and event level, may have an arbitrary number of hierarchically structured dimension levels.

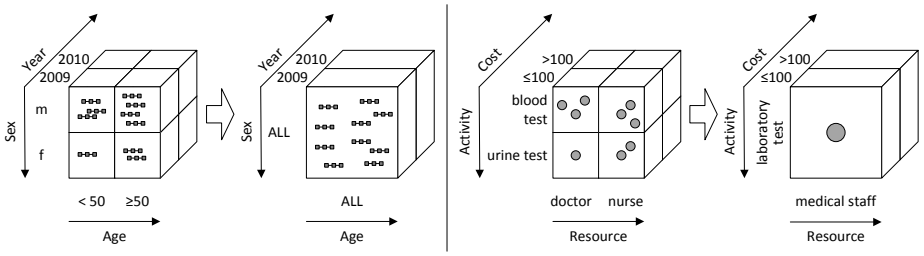


Fig. 6: Aggregation of cases (left) and events (right)

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 6 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 6 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 side of Figure 6, showing the aggregation of events along the dimensions *activity* and *resource*. Previously, various events are spread across different cells, 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 enables the analyst, 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. This may significantly speed up the OLAP query in case of slow database connections. 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 onto the relational data model which is discussed in the following section.

6 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.

Even though star schemes usually provide a better performance, we extend the traditional snowflake schema (cf. Section 2) to avoid redundancy which may lead to data anomalies. Figure 7 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

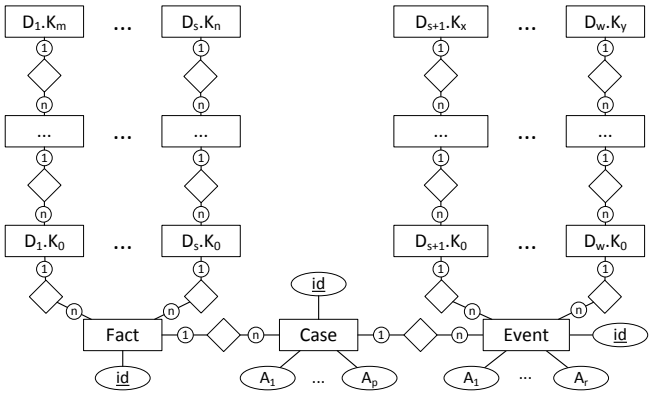


Fig. 7: Generic database schema of the MEL

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.

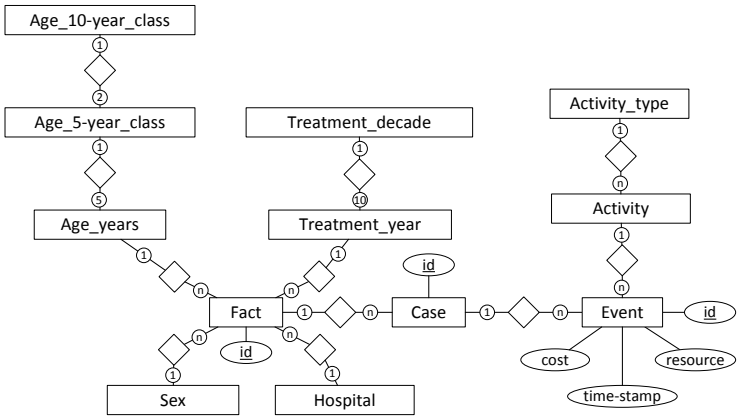


Fig. 8: Example database schema for the running example

Figure 8 shows a possible database schema for the running example as an entity-relationship model. All case attributes are modeled as dimensions. While the sex and hospital dimension only consist of a single dimension level, the age and the year of treatment dimensions have additional dimension levels. These

artificial dimension levels enable the analysts to group the patients by 5-year and 10-year classes respectively. For the events, only the activity attribute is mapped onto a dimension with two dimension levels. In this example, the *Activity_type* dimension level is introduced to define several types of activities, e.g., admission and discharge activities can be considered as organizational activities. All other event attributes (cost, time-stamp, and resource) are attached to the Event table as simple attributes. Note that there are also other possibilities to model the database schema for the running example, e.g., one could map the resource onto a dimension as well.

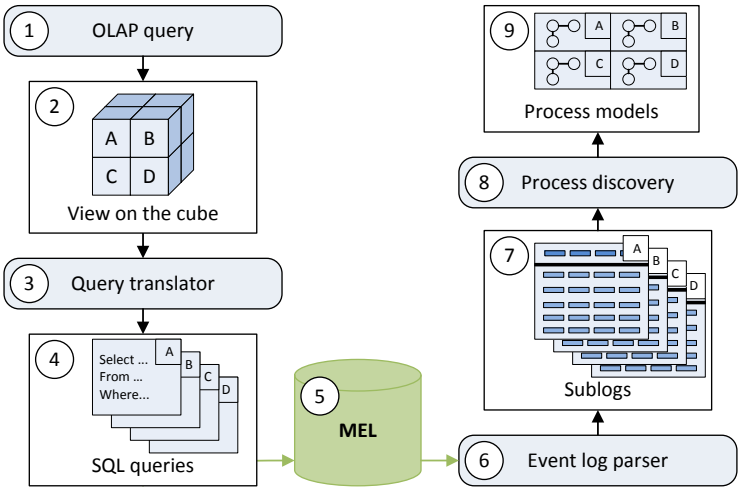


Fig. 9: Mapping an OLAP query to a set of SQL queries

Figure 9 illustrates how the event data is loaded from the MEL and processed in PMCube. The 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 7 presents the query translation in more detail. The SQL queries are sent to the MEL consisting of a relational database (cf. 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 [4] 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 time-stamp for this. However, it might be possible that the time-stamp 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.

7 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, PMCube 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 onto correspondent SQL queries. In Section 7.1, we first explain the general structure of the generic query pattern. After that, we show its extensions for event-based case filtering and event aggregation in Sections 7.2–7.5.

7.1 General Structure

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.

```
1  SELECT Event.case_id, Event.Timestamp, Activity.value,
2      Hospital.value, Sex.value, Treatment_year.value,
3          Event.resource, Event.cost
4  FROM Fact
5      JOIN Case ON Case.fact_id = Fact.id
6      JOIN Event ON Case.id = Event.case_id
7      JOIN Hospital ON Hospital.id = Fact.Hospital_id
8      JOIN Sex ON Sex.id = Fact.Sex_id
9      JOIN Year_of_treatment ON
10         Year_of_treatment.id = Fact.year_of_treatment_id
11         JOIN Activity ON Activity.id = Event.Activity_id
12 WHERE Hospital.value = 'Oldenburg'
13     AND Sex.value = 'female'
14     AND Year_of_treatment.value IN (2012, 2013, 2014)
15 ORDER BY Event.case_id, Event.timestamp
```

Listing 1.2: Example instantiation of the generic query pattern

Listing 1.2 shows an example instantiation of the generic query pattern based on the running example. Besides the case id, the time-stamp, and the event's activity, it also loads the name of the treating hospital, the sex of the patient, the year when the patient was treated, the resource performing the activity, and

the individual costs related to an event (lines 1-3). In addition to the mandatory join of the fact, case and event tables (lines 4-6), this query joins the fact and event tables with the dimension tables (lines 7-11) in order to retrieve the selected attributes from the dimension tables. As the query is only defined for a single cell, the WHERE clause restricts the events to all female patients that were treated in the hospital in Oldenburg (lines 12-13). Additionally, the data is filtered by the year of the treatment, restricting the result to treatments in the years 2012 to 2014 (line 14). Finally, the events of the result set are ordered by their case and the time-stamp to reconstruct the traces of the event log (line 15).

case id	**timestamp**	**activity**	**hospital**	**sex**	**year**	**resource**	**cost**
1	01.02.2012 08:30	Admission	Oldenburg	female	2012	nurse	58
1	01.02.2012 09:42	Examination	Oldenburg	female	2012	doctor	161
1	01.02.2012 10:38	Blood test	Oldenburg	female	2012	nurse	142
1	01.02.2012 10:51	X-ray scan	Oldenburg	female	2012	doctor	61
1	02.02.2012 09:11	Surgery	Oldenburg	female	2012	doctor	66
1	03.02.2012 08:23	Surgery	Oldenburg	female	2012	doctor	50
1	04.02.2012 13:27	Discharge	Oldenburg	female	2012	nurse	137
3	01.02.2014 08:30	Admission	Oldenburg	female	2014	nurse	102
3	01.02.2014 09:42	Examination	Oldenburg	female	2014	doctor	163
3	01.02.2014 10:38	Blood test	Oldenburg	female	2014	nurse	168
3	01.02.2014 10:51	X-ray scan	Oldenburg	female	2014	doctor	168
3	02.02.2014 09:11	Surgery	Oldenburg	female	2014	doctor	140
3	04.02.2014 13:27	Discharge	Oldenburg	female	2014	nurse	98
11	01.02.2013 08:30	Admission	Oldenburg	female	2013	nurse	67
11	01.02.2013 09:42	Examination	Oldenburg	female	2013	doctor	41
11	01.02.2013 10:38	X-ray scan	Oldenburg	female	2013	doctor	178
11	01.02.2013 10:51	Urine test	Oldenburg	female	2013	nurse	176
11	02.02.2013 09:11	Surgery	Oldenburg	female	2013	doctor	105
11	03.02.2013 08:23	Surgery	Oldenburg	female	2013	doctor	40
11	04.02.2013 13:27	Discharge	Oldenburg	female	2013	nurse	157

Table 6: Result of example query from Listing 1.2

Table 6 shows the query result when executing the query from Listing 1.2 on the event log L_1 (cf. Table 1). Due to the filter condition (only female patients who were treated in Oldenburg between 2012 and 2014), there are only three out of 20 cases selected. Each one of the resulting tuples represents a single event. The events are grouped by their related case and sequentially ordered by their case id and time-stamp. According to the projection (Listing 1.2, lines 1-2), the extracted data is restricted to a subset of the available attributes.

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.

7.2 Case Filter Extension: ∃ Subquery

Listing 1.3 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. Usually, removing duplicates is a very expensive database operation. However, this overhead can be significantly reduced by defining an index on the `case_id` attribute in the database.

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

Listing 1.3: Subquery for selecting cases with at least one event matching a condition

Based on the running example, Listing 1.4 shows an example subquery that selects all cases for which at least one blood test has been performed. All other cases are omitted from the query result. For convenience, we use readable and meaningful string ids for the activity in this example.

```
1  ... AND case_id IN (
2              SELECT UNIQUE case_id FROM Event
3              WHERE Activity_id = 'blood test'
4      ) ...
```

Listing 1.4: Example subquery for selecting cases with at least one blood test event

Assuming that the WHERE clause in the example of the generic query pattern (Listing 1.2) is extended by the condition shown in Listing 1.4, the query will return a subset of the result from Table 6 which only comprises the tuples related to cases 1 and 3. The tuples for case 11 will be removed from the result, because no blood test was conducted for this patient.

7.3 Case Filter Extension: ∀ Subquery

If the condition must hold for each event of a case, the subquery shown in Listing 1.5 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 1.5: Subquery for selecting cases where each event of a case matches a condition

Listing 1.6 shows an example subquery for selecting cases whose events were all executed by a nurse. As described above, the subquery selects all ids of all cases that have an event violating this condition, e.g., if an activity was performed by a doctor. Note that the attribute `resource` has to be checked for `NULL` values as events with an unknown resource violate the condition, too.

```
1  ... AND case_id NOT IN (
2           SELECT UNIQUE case_id FROM Event
3           WHERE NOT resource = 'nurse'
4               OR resource IS NULL
5  ) ...
```

Listing 1.6: Example subquery for selecting cases whose events were all executed by a nurse

Adding the subquery in Listing 1.6 to the WHERE clause of the generic query pattern example (Listing 1.2), will return an empty result set when executed on the example data from event log L_1. This is obvious as every patient already selected by the conditions defined in Listing 1.2 (female, treated between 2012 and 2014 in Oldenburg) has multiple events that where conducted by a doctor. Consequently, there is no case having all its events exclusively executed by nurses.

7.4 Case Filter Extension: Aggregation Subquery

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 1.7 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.

```
1 ... AND case_id IN (
2             SELECT UNIQUE case_id FROM Event
3             GROUP BY case_id
4             HAVING <condition>
5     ) ...
```

Listing 1.7: Subquery for selecting cases using aggregations over event attributes

An example for selecting cases by aggregated event attribute values is given in Listing 1.8. It selects all cases whose events in average exceed the cost limit of $100 per event.

```
1 ... AND case_id IN (
2         SELECT UNIQUE case_id FROM Event
3         GROUP BY case_id
4         HAVING AVG (cost) > 100
5 ) ...
```

Listing 1.8: Example subquery for selecting cases whose events have an average cost of more than $100

Extending the WHERE clause of the example query in Listing 1.2 by the subquery from Listing 1.8 and executing it on event log $L1$ will return a subset of Table 6. The query result will only contain the events for the cases 3 and 11 as their average costs ($139.8 and $109.1, respectively) exceed the threshold of $100. The events related to case 1 will be removed because the average event costs of 96.4$ do not meet the condition.

7.5 Event Aggregation Extension

Finally, it is also possible to realize the aggregation of events (cf. Section 5) within the SQL query. For this operation, we extend the generic query pattern 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, Activity.value` 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 a 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.

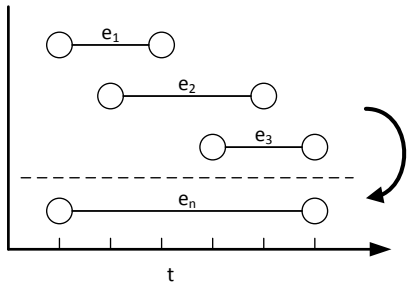


Fig. 10: Aggregating start and end time-stamps of events

Figure 10 illustrates the merging of the start and end time-stamps of events as another example. We use the MIN function for the start time-stamp and the MAX function for the end time-stamp. Consequently, 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.

Listing 1.9 shows an extended query based on the example query from Listing 1.2 that aggregates the events by the activity. This results in a sublog where each activity occurs at most once per trace, e.g., each trace has at most one *blood test* activity. In this example, we use the MIN function to aggregate the values of the hospital, the sex, the year of treatment dimensions because SQL only allows to select aggregated values or grouping attributes. As all case attributes have the same value for all events of a case, selecting the minimum does not change the results here. For the event attribute, we use the MIN function as well while we use the SUM function to calculate for the event costs. In contrast to the example presented in Figure 10, we only have a single time-stamp related to an event. Therefore, we map the time-stamps of the aggregated events onto its earliest occurrence using the minimum. However, this is a subjective choice. Alternatively it is also possible to use other aggregation functions, e.g., the

```
1  SELECT Event.case_id, min(Event.Timestamp) as alias_ts,
2         Activity.value, min(Hospital.value),
3         min(Sex.value), min(Treatment_year.value),
4         min(Event.resource), SUM(Event.cost)
5  FROM Fact
6      JOIN Case ON Case.fact_id = Fact.id
7      JOIN Event ON Case.id = Event.case_id
8      JOIN Hospital ON Hospital.id = Fact.Hospital_id
9      JOIN Sex ON Sex.id = Fact.Sex_id
10     JOIN Year_of_treatment ON
11         Year_of_treatment.id =Fact.year_of_treatment_id
12         JOIN Activity ON Activity.id = Event.Activity_id
13 WHERE Hospital.value = 'Oldenburg'
14     AND Sex.value = 'female'
15     AND Year_of_treatment.value IN (2012, 2013, 2014)
16 GROUP BY Event.case_id, Activity.value
17 ORDER BY Event.case_id, alias_ts
```

Listing 1.9: Extended example query aggregating events by their activity

maximum to map the events onto the latest occurrence. Note that the case id and the activity dimension do not require an aggregation function because both are used in the GROUP BY clause (line 16). Furthermore, the aggregated time-stamp is renamed to `alias_ts` in order to use the aggregated value for ordering the events.

Table 7 shows the result for executing the query from Listing 1.9 on the example data of Event Log L_1. Its content is quite similar to the results shown in Table 6. The only difference in this example is that the surgery events for cases 1 and 11 are aggregated into a single event for each case (highlighted in Table 7). For the merged event, the time-stamp of the first event is used. For the cost attribute, the individual costs of the original events are summed up.

The case study reported in [18] revealed that the aggregation of events is able to improve the fitness of the resulting process models if the frequency of an activity per case does not correlate with its importance. For example, each parameter of a blood test event may be represented by an individual event. Algorithms which consider the ratio between the events, tend to overestimate the importance of these events. The Fuzzy Miner [5], for example, will cluster the less frequent events into one node while representing the more frequent but less important blood test events as individual nodes. Aggregating the events by their activity removes this bias because it restricts the maximum frequency per trace of each activity to one. On the contrary, the process model can also be significantly biased by the event aggregation itself. Especially if other events are located between the aggregated events in the trace, the dependencies to the aggregated events are lost. This may be indicated by a significant loss of fitness of the resulting process model. Therefore, the analysts are recommended to be aware of the possible bias of the process model and apply the event aggregation carefully.

case id	timestamp	activity	hospital	sex	year	resource	cost
1	01.02.2012 08:30	Admission	Oldenburg	female	2012	nurse	58
1	01.02.2012 09:42	Examination	Oldenburg	female	2012	doctor	161
1	01.02.2012 10:38	Blood test	Oldenburg	female	2012	nurse	142
1	01.02.2012 10:51	X-ray scan	Oldenburg	female	2012	doctor	61
1	**02.02.2012 09:11**	**Surgery**	**Oldenburg**	**female**	**2012**	**doctor**	**116**
1	04.02.2012 13:27	Discharge	Oldenburg	female	2012	nurse	137
3	01.02.2014 08:30	Admission	Oldenburg	female	2014	nurse	102
3	01.02.2014 09:42	Examination	Oldenburg	female	2014	doctor	163
3	01.02.2014 10:38	Blood test	Oldenburg	female	2014	nurse	168
3	01.02.2014 10:51	X-ray scan	Oldenburg	female	2014	doctor	168
3	02.02.2014 09:11	Surgery	Oldenburg	female	2014	doctor	140
3	04.02.2014 13:27	Discharge	Oldenburg	female	2014	nurse	98
11	01.02.2013 08:30	Admission	Oldenburg	female	2013	nurse	67
11	01.02.2013 09:42	Examination	Oldenburg	female	2013	doctor	41
11	01.02.2013 10:38	X-ray scan	Oldenburg	female	2013	doctor	178
11	01.02.2013 10:51	Urine test	Oldenburg	female	2013	nurse	176
11	**02.02.2013 09:11**	**Surgery**	**Oldenburg**	**female**	**2013**	**doctor**	**145**
11	04.02.2013 13:27	Discharge	Oldenburg	female	2013	nurse	157

Table 7: Result of example query from Listing 1.9 (event aggregation)

8 Evaluation

We implemented our approach in a prototype called PMCube Explorer [17]. Due to a generic, plug-in-based interface, arbitrary DBMS 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 [18]. 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 and Appelrath [18]. We created multiple subsets of that data, each consisting of a particular number of events. While creating these subsets, we kept the cases as a whole in order to avoid splitting up the traces into fragments. Table 8 shows the number of cases and events for each data set. Figure 11 clearly shows the linear relationship between the number of cases and the number of events of the data sets.

Sample	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Events	50,000	100,000	150,000	200,000	250,000	300,000	350,000	388,395
Cases	2,154	4,332	6,340	8,448	10,418	12,538	14,734	16,280

Table 8: Number of cases and events of the evaluation's data sets

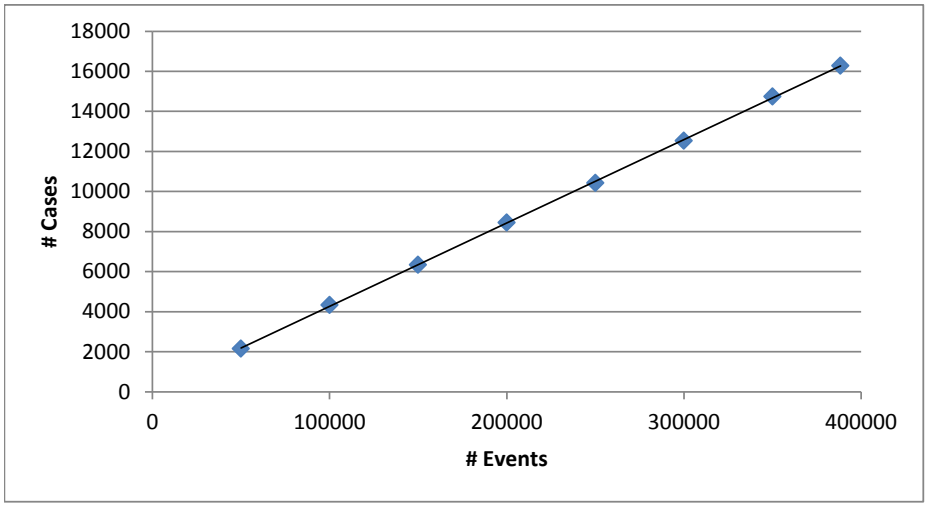


Fig. 11: Linear relationship of the number of cases and events for the evaluation's subsets of data

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. This query results in a cube with 12 cells.

Q2: The dimensions *medical department* and *reason for discharge* form the cube. No filtering was used. Depending on the used subset of the data, this query creates 40 to 56 cells for PMC which only considers dimension values reflected in the data set. In contrast, PMCube Explorer considers each possible dimension value which is defined in the database, whether it is reflected by the events or not. Therefore, it procudes 92 cells for each subset.

Q3: The dimensions *age* (in 10-year-classes) and *sex* form the cube. All cases with values *unknown* and *older than 100 years* were removed. This query results in 30 cells.

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. The resulting data cube for this query contains six cells.

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

Both tools, PMC and PMCube Explorer, vary in the set of supported operations. For example, PMCube Explorer provides the aggregation of events while PMC does not support this. Consequently, we only used queries that could be defined and answered by both tools. All queries partition the data by a number of case attributes into sublogs and filter the data by neglecting particular cells. As the definition of these operations is similar for both approaches, the resulting sublogs are congruent. However, PMCube Explorer may return more cells than PMC (cf. query Q2) as it also considers predefined dimension values that are not represented in the data. However, these additional cells do not affect the results of other cells and only require minimal overhead because they only contain empty sublogs.

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

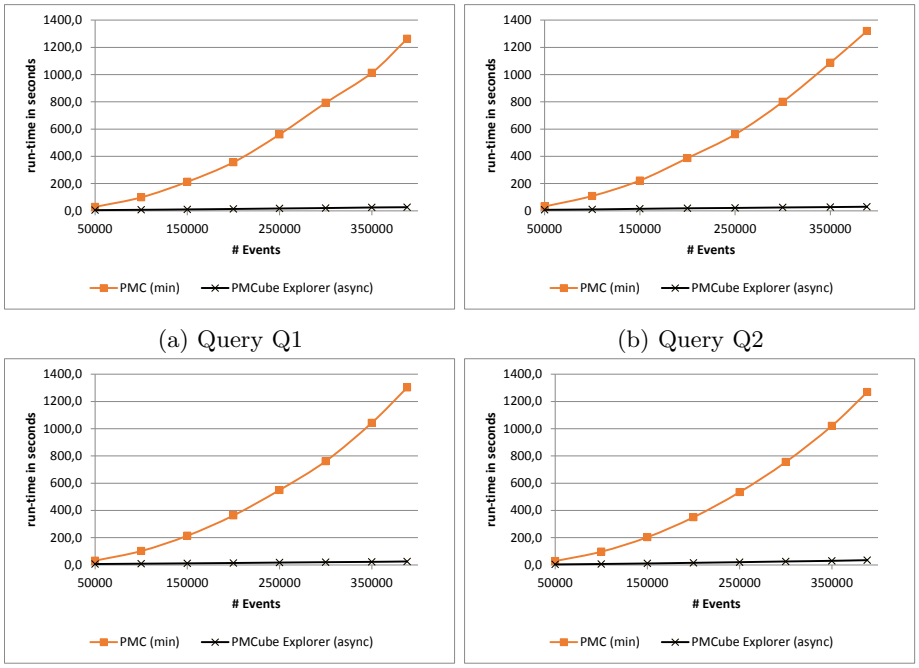
Table 9: Average run-times in seconds

Reflecting the overall waiting time for the analysts, we measured the total run-time for the processing of the OLAP query, discovering the model using Inductive Miner [6], 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 from the data cube which were not used to express the OLAP query. However, for our approach PMCube, we kept all available dimensions in the data cube. Even though this might overrate the performance of PMC, the affect on run-time should be insignificant because dimension tables are not joined unless they are used in the OLAP query.

Table 9 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, i.e., the processing of a cell is completed before the next cell

is loaded from the database. Note that the last column of Table 9 shows the measured values for the full data set.

The values in Table 9 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.



(a) Query Q1

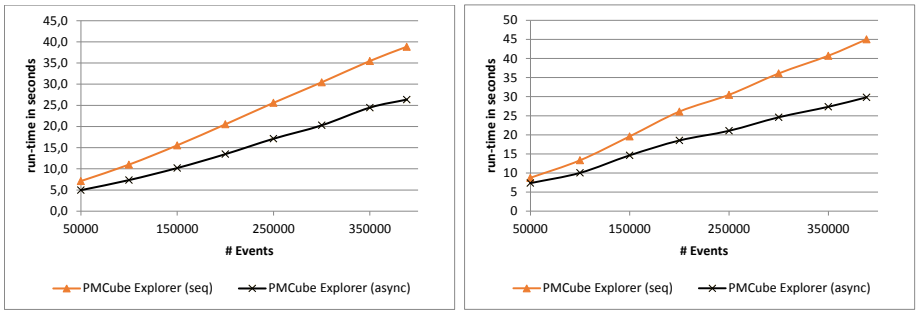
(b) Query Q2

(c) Query Q3

(d) Query Q4

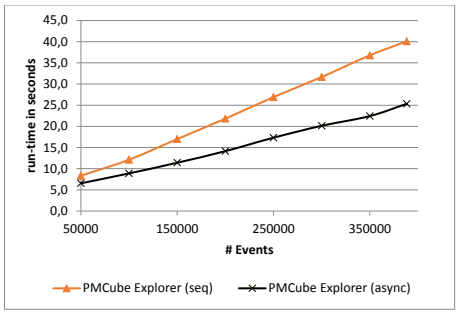
Fig. 12: Comparing average run-time of PMCube Explorer (async) with PMC (min) for queries Q1-Q4

Figure 12 shows the run-times (in seconds) of the queries Q1-Q4 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. Comparing the four charts to each other, PMC as well as PMCube Explorer show a similar run-time behavior for all queries. Figure 13 compares the run-time of both configurations of PMCube Explorer for the queries Q1-Q4. 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, even though the advantage of the asynchronous processing varies for each query.

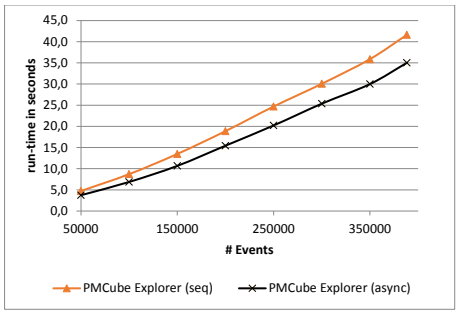


(a) Query Q1

(b) Query Q2



(c) Query Q3



(d) Query Q4

Fig. 13: Comparing average run-time of PMCube Explorer (async) with PMCube Explorer (seq) for queries Q1-Q4

Table 10 shows the measured loading times in seconds for PMC and both configurations of PMCube Explorer. The measured values reflect the time needed for processing the query and returning a set of sublogs. Comparing the different approaches clearly shows that the loading times of PMCube Explorer are significantly lower than the loading times of PMC. However, the loading times for PMCube Explorer (seq) are shorter than for PMCube Explorer (async). At first glance, this is in contrast to the overall processing times shown in Table 9, which are shorter for the asynchronous processing. These differences can be explained by the additional overhead required for the asynchronous processing. In the evaluation setting, all processes – also the database management system – are running on the same machine in parallel. Consequently, the loading of data (including the parsing of SQL results), the discovery of processes (implemented as an individual process for each cell) and the data processing of the database are competing for the same resources, especially CPU. Therefore, the loading process of PMCube Explorer may be waiting for resources while previously started process discovery threads are occupying the CPU cores. Due to this waiting times and the additional effort for process switches, the loading times are longer in case of asynchronous processing. However, as they significantly reduce the

Query	Events (in thousand)	50	100	150	200	250	300	350	388.395
Q1	PMC (min)	27.9	97.4	211.6	354.8	558.7	790.8	1010.6	1260.7
	PMCube Explorer (seq)	1.9	3.4	4.9	6.6	8.0	9.7	11.5	12.7
	PMCube Explorer (async)	2.5	4.6	6.7	9.0	11.0	13.1	15.2	17.8
Q2	PMC (min)	30.5	107.1	220.4	385.0	558.4	797.4	1083.8	1316.4
	PMCube Explorer (seq)	2.9	4.7	6.6	8.8	10.7	12.8	15.2	16.6
	PMCube Explorer (async)	4.0	6.3	9.7	12.6	16.0	19.1	21.8	24.2
Q3	PMC (min)	29.8	100.2	212.5	361.5	547.4	759.1	1040.5	1299.8
	PMCube Explorer (seq)	2.4	4.1	5.8	7.7	9.4	11.2	13.9	14.5
	PMCube Explorer (async)	3.1	5.6	8.0	10.6	13.2	15.9	18.4	21.2
Q4	PMC (min)	27.4	96.0	202.4	348.8	532.6	753.7	1018.8	1267.1
	PMCube Explorer (seq)	1.6	3.0	4.4	5.9	7.2	8.5	10.2	11.4
	PMCube Explorer (async)	1.7	3.2	4.7	6.3	7.9	9.4	11.0	12.1

Table 10: Average loading times in seconds

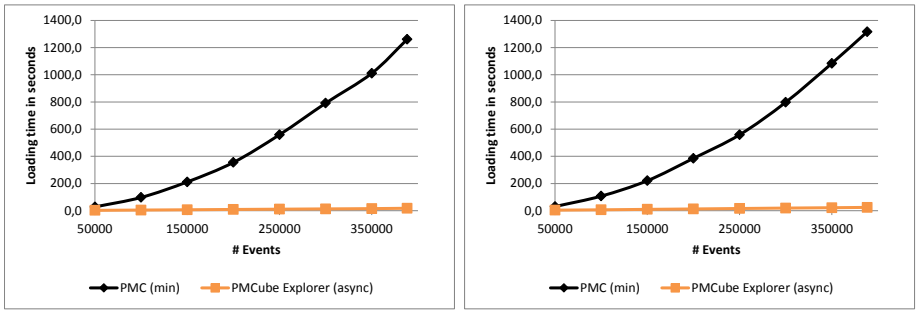
overall run-time for processing a query, these delays while loading the data are justified.

Figure 14 shows the average loading times in seconds over the number of events for PMC and PMCube Explorer (async) for queries Q1-Q4. The charts reveal the same behavior as the overall run-time depicted in Figure 12. While the loading time grows polynomially with the number of events for PMC, it grows linearly for PMCube Explorer (async). This clearly shows that the PMCube Explorer's advantage in run-time can be traced back to the data storage and management based on the relational data warehouse. The charts presented in Figure 15 show the average loading times in seconds over the number of events for both configurations of PMCube Explorer for queries Q1-Q4. They also confirm the linear incline of the loading time by the number of events. Furthermore, the charts indicate that the measured loading times of PMCube Exporer (async) are bigger than the loading times of PMCube Explorer (seq). However, comparing the charts also shows that the difference varies between the queries. While the difference between both configurations is significant for queries Q1-Q3, there is only a slight difference for Q4.

Due to the clear linear relationship between the number of events and the number cases of the evaluation's data sets (cf. Table 8 and Figure 11), the observed run-time behavior can also be related to the number cases. Consequently, the run-times increases polynomially by the number of cases for PMC, while it increases linearly for PMCube Explorer.

9 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



(a) Query Q1

(b) Query Q2

(c) Query Q3

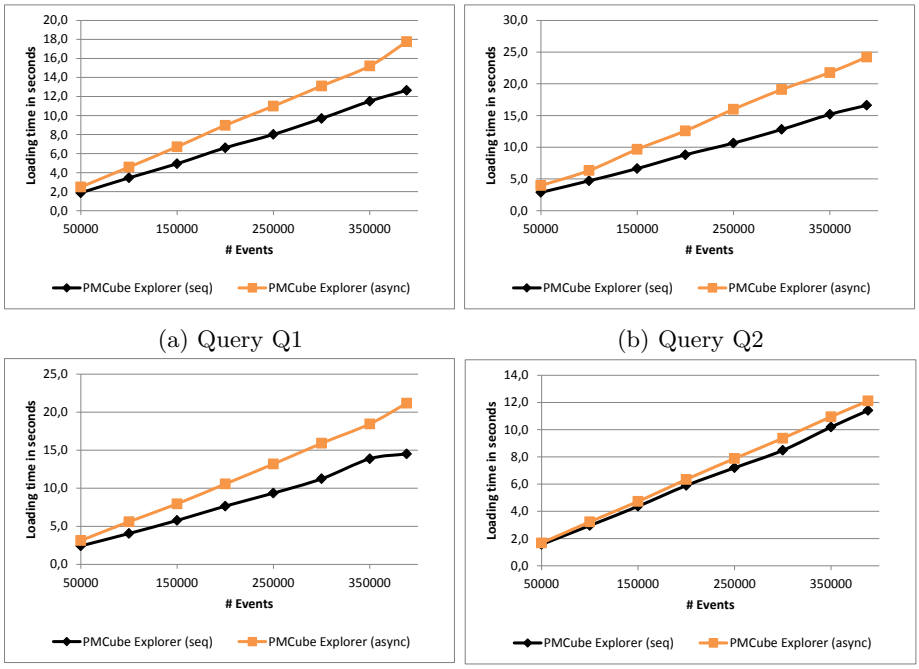
(d) Query Q4

Fig. 14: Comparing average loading time of PMCube Explorer (async) with PMC (min) for queries Q1-Q4

mapped onto a generic relational database schema. We use generic query 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. Furthermore, it should be investigated how to improve the flexibility of our approach, e.g., how to reflect multiple case ids in the database schema without loosing performance.

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)



(a) Query Q1 (b) Query Q2

(c) Query Q3 (d) Query Q4

Fig. 15: Comparing average loading time of PMCube Explorer (async) with PMCube Explorer (seq) for queries Q1-Q4

2. Ekanayake, C.C., Dumas, M., García-Bañuelos, L., La Rosa, M.: Slice, Mine and Dice: Complexity-Aware Automated Discovery of Business Process Models, pp. 49–64. Springer Berlin Heidelberg, Berlin, Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-40176-3_6
3. Golfarelli, M., Rizzi, S.: Data Warehouse Design: Modern Principles and Methodologies. McGraw-Hill, Inc., New York, NY, USA, 1 edn. (2009)
4. Günther, C.W.: XES Standard Definition (Mar 2014), http://www.xes-standard.org/xesstandarddefinition
5. Günther, C.W., van der Aalst, W.M.P.: Fuzzy mining: adaptive process simplification based on multi-perspective metrics. In: Proceedings of the 5th international conference on Business process management. pp. 328–343. BPM'07, Springer-Verlag, Berlin, Heidelberg (2007)
6. 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)
7. Maggi, F.M., Dumas, M., García-Bañuelos, L., Montali, M.: Discovering Data-Aware Declarative Process Models from Event Logs, pp. 81–96. Springer Berlin Heidelberg, Berlin, Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-40176-3_8

8. 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)
9. 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)
10. Nooijen, E.H.J., van Dongen, B.F., Fahland, D.: Automatic discovery of data-centric and artifact-centric processes. In: Business Process Management Workshops - BPM 2012 International Workshops, Tallinn, Estonia, September 3, 2012. Revised Papers. pp. 316–327 (2012), http://dx.doi.org/10.1007/978-3-642-36285-9_36
11. 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)
12. Schönig, S., Rogge-Solti, A., Cabanillas, C., Jablonski, S., Mendling, J.: Efficient and Customisable Declarative Process Mining with SQL, pp. 290–305. Springer International Publishing, Cham (2016), http://dx.doi.org/10.1007/978-3-319-39696-5_18
13. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer, Berlin (2011)
14. 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)
15. 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)
16. van Dongen, B.F., Shabani, S.: Relational XES: Data Management for Process Mining. In: Proceedings of the CAiSE 2015 Forum at the 27th International Conference on Advanced Information Systems Engineering co-located with 27th International Conference on Advanced Information Systems Engineering (CAiSE 2015), Stockholm, Sweden, June 10th, 2015. pp. 169–176 (2015), http://ceur-ws.org/Vol-1367/paper-22.pdf
17. 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
18. Vogelgesang, T., Appelrath, H.J.: PMCube: A Data-Warehouse-Based Approach for Multidimensional Process Mining. In: Reichert, M., Reijers, A.H. (eds.) Business Process Management Workshops: BPM 2015, 13th International Workshops, Innsbruck, Austria, August 31 – September 3, 2015, Revised Papers. pp. 167–178. Springer International Publishing (2016), http://dx.doi.org/10.1007/978-3-319-42887-1_14