Configuring SQL-based Process Mining for Performance and Storage Optimisation

Stefan Schönig
University of Bayreuth
Bayreuth, Germany
stefan.schoenig@uni-bayreuth.de

Claudio Di Ciccio
Vienna University of Economics and Business
Vienna, Austria
claudio.di.ciccio@wu.ac.at

Jan Mendling
Vienna University of Economics and Business
Vienna, Austria
jan.mendling@wu.ac.at

ABSTRACT

Process mining is the area of research that embraces the automated discovery, conformance checking and enhancement of process models. Declarative process mining approaches offer capabilities to automatically discover models of flexible processes from event logs. However, they often suffer from performance issues with real-life event logs, especially when constraints to be discovered go beyond a standard repertoire of templates. By leveraging relational database performance technology, a new approach based on SQL querying has been recently introduced, to improve performance though still keeping the nature of discovered constraints customisable. In this paper, we provide an in-depth analysis of configuration parameters that allow for a speed-up of the answering time and a decrease of storage space needed for query processing. Thereupon, we provide configuration recommendations for process mining with SQL on relational databases.

CCS CONCEPTS

• **Information systems → Information extraction**; • **Theory of computation → Modal and temporal logics**; *Logic and databases*; • **Applied computing → Business process modeling**; **Business intelligence**; *Business process management systems*; *Business rules*;

KEYWORDS

Declarative process mining, relational databases, SQL

ACM Reference Format:
Stefan Schönig, Claudio Di Ciccio, and Jan Mendling. 2019. Configuring SQL-based Process Mining for Performance and Storage Optimisation. In *The 34th ACM/SIGAPP Symposium on Applied Computing (SAC '19), April 8–12, 2019, Limassol, Cyprus.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3297280.3297532

1 INTRODUCTION

Declarative process modelling languages like *Declare* [10] or *Declarative Process Intermediate Language* (DPIL) [16] can be used to represent flexible processes, and tools like MINERful [5] or DPILMiner [12] offer capabilities to automatically discover such models from event logs. Existing declarative process mining approaches either suffer from performance issues with real-life event logs or limit their search space to a specific and fixed set of constaints to be able to cope with the size of real-life event logs. Both issues are highlighted in current literature [5]. Recently, a process mining approach based on SQL that works on event data that is stored in relational databases has been introduced [13, 14]. Process mining by means of SQL turned out to be a fast and customisable solution to process discovery by leveraging relational database performance technology, e.g., indexes on data columns. While existing work explains how to extract process constraints by SQL queries, it remains unclear how the underlying relational database needs to be configured to optimise querying and to minimise storage requirements. With this paper, we further complete the research on SQL-based process mining by providing an in-depth analysis of database configuration parameters and their influence on query performance and storage requirements. By analysing the effect of each parameter combination on query processing w.r.t. different real-life event logs, the work at hand provides a configuration recommendation for process mining with SQL on relational databases. This paper is structured as follows: Section 2 describes the input data and the fundamentals of declarative process mining with SQL. In Section 3 we describe the possible configuration parameters of SQL-based process mining and the performance and storage measurement results. The results are discussed in Section 4, and Section 5 concludes the paper.

2 BACKGROUND AND RELATED WORK

Declarative process mining techniques check the satisfaction of constraints based on the metrics *Support* and *Confidence*. *Response(a,b)* is a constraint of the Declare language on activities *a* and *b*, forcing *b* to be executed eventually if activity *a* was performed before. The *ChainResponse(a,b)* constraint forces activity *b* to be executed *directly* after activity *a* was executed. Constraints are considered valid if their Support and Confidence measures are above a user defined threshold. Several declarative mining approaches for Declare have been presented, a.o. [7, 9]. In [1], the authors present an approach for the mining of declarative processes expressed through probabilistic logics. A sequence-analysis based algorithm for Declare discovery is proposed in [6], later improved with parallel processing in [8]. The *MINERful* approach [5] is to date the most efficient algorithm to discover control-flow constraints. Beyond Declare, an algorithm for discovering DCR graphs has been recently presented in [2]. The *DPILMiner* [12] proposes an approach to incorporate the resource perspective. In [11] the authors present a declarative approach to discover resource team compositions from process event logs. Post-processing approaches have been proposed that

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
SAC '19, April 8–12, 2019, Limassol, Cyprus
© 2019 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5933-7/19/04.
https://doi.org/10.1145/3297280.3297532

aim at simplifying the resulting declarative models in terms of, a.o., inconsistency elimination [3] and redundancy reduction [4, 5, 7]. All the mentioned approaches, however, suffer from performance issues w.r.t. real-life event logs, especially when they comprise an elevated number of activities. Lately, with *RelationalXES (RXES)* a relational database architecture for storing event log data has been introduced [15]. The RXES architecture uses a database to store the event log where traces and events are represented by tables with identifiers (IDs). The database schema used in RXES allows for a significant reduction of redundancy by storing frequently occurring attributes only once rather than repeating them for every occurrence. In [14] we showed that it is possible to discover constraints by means of conventional SQL queries.

3 ANALYSIS OF REQUIREMENTS

In this section, we describe the configuration and show the results of measuring the performance and storage requirements of SQL-based mining. In all well-known relational databases performance of query processing and the storage requirements of the data can be influenced by different set screws. We identified three parameters whose influence on performance and storage will be analysed: *(i) indexes (clustered or non-clustered), (ii) datatypes (integer or string)* and *(iii) parallelism (single-core or multi-core)*. We show the results of measuring the performance and storage requirements of SQL-based mining w.r.t. two real-life event logs *(i)* from the financial[1] and *(ii)* healthcare[2] domains by means of queries for the *Response*- and the *ChainResponse*-constraints. We evaluate our approach by means of these two logs because they have fundamental differences: log *(i)* contains 262000 events of 13087 traces refering to 24 activities. Log *(ii)* on the other hand contains 150291 events of 1143 traces refering to 624 activities. We especially want to draw attention to the big difference in the number of activities (624 to 24). The logs have been imported into a Microsoft SQL Server 2014 database both with data as integers[3] as well as as strings. All the computation times reported in this section are measured on Quad-Core i7 CPUs @2.80 GHz with 8 GB RAM.

3.1 Performance Analysis

First, we measured the performance of queries on both logs w.r.t. different configurations. Alongside with the properties of the database, i.e., configuration of indexes (*none, clustered, non-clustered, both*) and datatypes of the log columns (*integer, string*), we made separate measurements for *multi-core* and *single-core* executions, marked with (m) and (s), respectively. The measurements in this section provide performance values for each combination of parameters. All time values are given in (min:sec). In each table the column with grey backcolor highlights the configuration with the best performance for each log. Table 1 shows the results of applying the *Response*-query to both logs with different configurations. It is remarkable that the execution time is in any configuration considerably higher for the hospital log than for the financial log. For the financial log the multi-core processing with both clustered and non-clustered indexes generated depicts the best performance. On the contrary, for the hospital log a single-core execution represents the most efficient configuration. Note, that in case of the hospital event log the single-core execution outperforms the multi-core execution in every conducted measurement. For both event logs the generation of indexes leads to a considerable improvement of performance, e.g., from 2:28 without index to 1:33 with both indexes for single-core processing on the hospital event log. However, the performance improvement of non-clustered indexes exceeds clustered indexes significantly in every conducted measurement. Remarkable is also the influence of the column datatypes on the performance w.r.t. the hospital log. Here, query processing on integer values is clearly faster than on string values, e.g., 1:33 to 0:16 with both indexes for single-core processing.

Table 1: Performance Measurement: *Response*-query

	None	Clustered	Non-Clustered	C & NC
	Financial Log			
Int.	0:05 (m);0:07 (s)	0:04 (m);0:06(s)	0:04 (m);0:06 (s)	0:03 (m);0:05 (s)
Str.	0:05 (m);0:07 (s)	0:04 (m);0:06 (s)	0:04 (m);0:06 (s)	0:03 (m);0:05 (s)
	Hospital Log			
Int.	1:46 (m);1:24 (s)	0:53 (m);0:23(s)	0:42 (m);0:18 (s)	0:41 (m);0:16 (s)
Str.	2:48 (m);2:28 (s)	2:26 (m);2:11 (s)	1:58 (m);1:50 (s)	1:50 (m);1:33 (s)

Table 2: Performance Measurement: *ChainResponse*-query

	None	Clustered	Non-Clustered	C & NC
	Financial Log			
I	0:06 (m);0:11 (s)	0:06 (m);0:11(s)	0:06 (m);0:10 (s)	0:05 (m);0:09 (s)
S	0:09 (m);0:13 (s)	0:08 (m);0:12 (s)	0:08 (m);0:12 (s)	0:07 (m);0:11 (s)
	Hospital Log			
I	5:29 (m);7:23 (s)	5:21 (m);7:10 (s)	4:38 (m);5:47 (s)	4:20 (m);5:02 (s)
S	10:23(m);16:01(s)	10:04(m);15:34(s)	8:39 (m);11:11 (s)	9:37 (m);10:21(s)

Table 2 shows the results of applying the *ChainResponse*-query to both logs with different configurations. Similar to the *Response*-query, the performance considerably differs from the application to the financial log to the hospital log (0:05 to 4:20). In case of the *ChainResponse*-query the multi-core processing with both clustered and non-clustererd indexes depicts for both logs the best performance. For this query the multi-core processing outperforms the single-core executions in every conducted simulation. Similar to Table 1 the generation of indexes significantly improves the performance, e.g., from 5:29 without index to 4:20 with both indexes for multi-core processing on the hospital event log. In turn, the performance improvement of non-clustered indexes exceeds clustered indexes clearly. In case of the *ChainResponse*-query, the influence of the column datatypes on the performance is even bigger and becomes obvious for both event logs. Similarly, query processing on integer values is significantly faster than on string values, e.g., 9:37 to 4:20 with both indexes for multi-core processing on the hospital log and 0:07 to 0:05 on the financial log.

3.2 Storage Analysis

In addition to the query execution performance we measured the storage requirements of both events logs with different configurations, i.e., with integer-based as well as string-based data and with different indexes defined. Table 3 shows the results, where all values are given in megabyte. The table shows both the space required for the data (*DS*) as well as the space required for the indexes (*IS*).

[1] doi: 10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
[2] doi: 10.4121/d9769f3d-0ab0-4fb8-803b-0d1120ffcf54
[3] While importing, we used the integer hashcodes of instance and activity names provided by the Microsoft .NET framework getHashcode() function.

It is remarkable that both event log tables need significantly less space than the original XML-based XES-files, e.g., the string-based financial log takes only 8.5 MB space compared to 70.6 MB of the XML-file. In turn, the integer-based hospital log takes 8.36 MB compared to 81.8 MB of the XML-file. Defining indexes increases the overall storage requirements in every case. While clustered indexes require only very little space, non-clustered indexes require in both cases even more space than the actual data. Here, string-based indexes have the most extreme space requirement, e.g., 40.37 MB for both indexes defined on the hospital log. Indexes on integer-based columns require on both logs only half of the space, e.g., 15.49 MB for both indexes on the hospital log. Furthermore, the results show that the combination of clustered and non-clustered indexes leads to an index space increase of factor two.

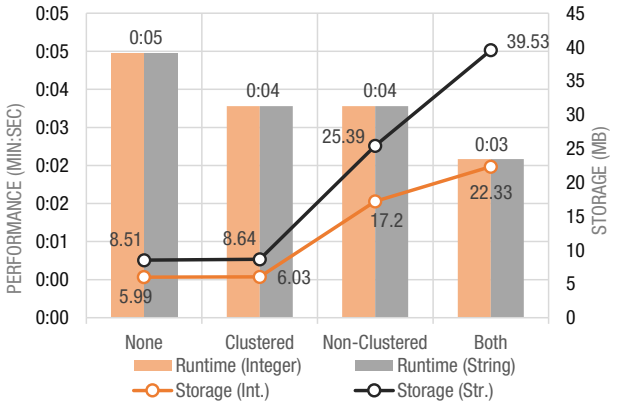
4 DISCUSSION AND FINDINGS

Here, we discuss the performance and storage measurement results. We first focus on the performance results that are explained by means of the diagrams in Fig. 1 and Fig. 2. Fig. 1a shows the execution time (bars) of the *Response*-query with different configurations on the financial log and Fig. 1b on the hospital log. The line chart in both diagrams show the storage requirements for the corresponding configuration. Figure 2a shows the performances of both the *Response*- and the *ChainResponse*-query on the financial log with multi-core and single-core execution and Fig. 2b on the right-hand-side shows the values for the hospital log. There are several observations that can be discovered:

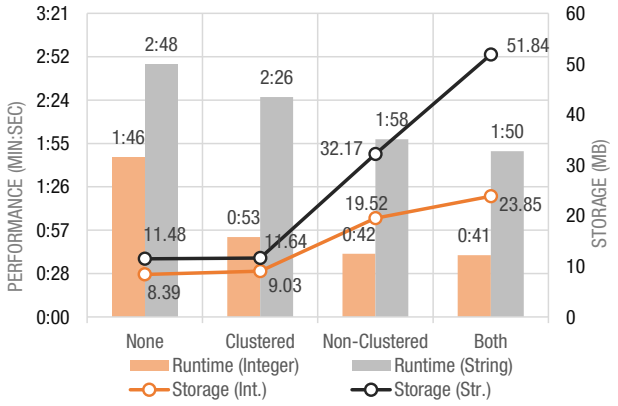
The first conclusion that can be drawn from both figures is that SQL-query runtime is determined by the number of distinct activities in the log and less by the number of events or traces. Comparing Fig. 2a and Fig. 2b, both the *Response*- and the *ChainResponse*-query need much more processing time on the hospital log that contains 600 activities more than the financial log. Independently from the type of event log, the query processing performance can be reduced by generating indexes. While the influence of clustered indexes is rather small, non-clustered indexes have a clear effect. The combination of clustered and non-clustered indexes provides an even better performance. While these observations can be made in almost every configuration, the clearest effect can be seen in case of the hospital log in Fig. 2b. Here, runtime is reduced from 1:24 (single-core, no index) to 0:16 (single-core, both indexes) and 7:23 (multi-core, no index) to 5:02 (multi-core, both indexes), respectively. In the first case, the execution with indexes only takes 19% of the original query execution time. Using integer-based data can reduce processing time significantly. While this effect does not show up in the financial log (Fig. 1a) it becomes crucial in the hospital log (Fig. 1b). Here, the execution time can be reduced from 1:50 to 0:41 for the *Response*-query (multi-core) and from 9:37 to 4:20 for *ChainResponse*-query

Table 3: Storage measurement (in MB)

	None	Clustered	Non-Clustered	C & NC
	Financial Log			
Integer	5.96 (DS); 0.03 (IS)	5.96; 0.07	5.96; 11.24	5.96; 16.37
String	8.50 (DS); 0.01 (IS)	8.50; 0.14	8.50; 16.89	8.50; 31.03
	Hospital Log			
Integer	8.36 (DS); 0.03 (IS)	8.36; 0.67	8.36; 11.16	8.36; 15.49
String	11.47 (DS); 0.01 (IS)	11.47; 0.17	11.47; 20.70	11.47; 40.37



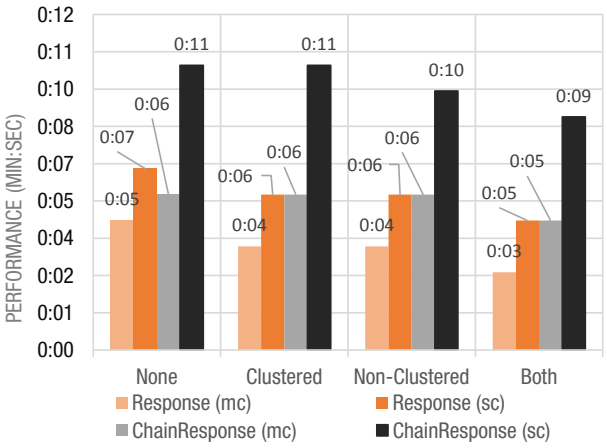
(a) Financial log (24 activities)



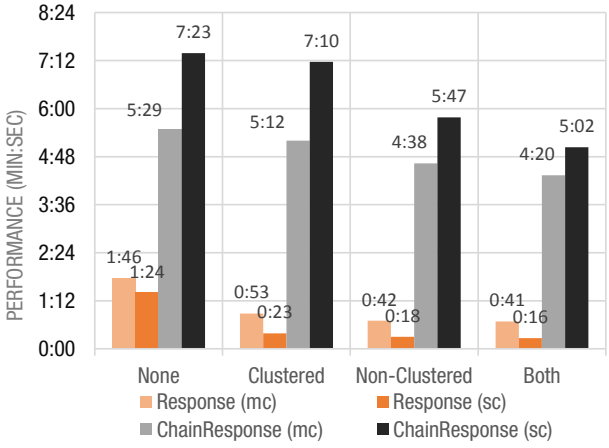
(b) Hospital log (624 activities)

Figure 1: *Response*-query with index settings

(multi-core). When analysing event logs with a huge number of activities, the pretransformation of strings to integer values (e.g. hash codes or incremental identifiers) has a big effect on execution. The results of analysing the effect of parallelisation are two-fold and can be seen in Fig. 2. Multi-core execution outperforms in every setting the single-core execution in case of the financial log that comprises a small number of activities. The situation differs in case of the hospital log that contains a big number of activities. Here, the *Response*-query is executed faster in a single-core setting than in a multi-core setting. In this case, the parallelisation overhead exceeds the advantage gained through parallising the query. This parallelisation overhead doesn't appear when querying the financial log. The more complex *ChainResponse*-query is executed clearly faster in a multi-core setting (5:02 to 4:20 for both indexes defined). In contrast to the traditional XML-based XES-format event log files the event log tables in the relational database have lower storage requirements. This becomes especially interesting when event data is recorded constantly for continous analysis. Performance improvement through index generation comes with its costs: the index space, i.e., the required disk space to store the index data structures exceed in some cases the space required for storing the



(a) Financial log (24 activities)



(b) Hospital log (624 activities)

Figure 2: *Response*-/*ChainResponse* on integer database

actual data by factor 2. The line charts in Fig. 1 show the overall storage requirements of the event log tables. With decreasing runtime the storage requirements for the database tables increase. It is additionally remarkable that the higher storage requirements of string-based data tables have a much bigger effect on the index space than on the data space.

Altogether, these results give us insights into how SQL-based process mining should be configured: *(i)* create both clustered and non-clustered indexes. The non-clustered indexes should be defined separately for the event log columns *EventID*, *Time*, *Activity* and *Instance*; *(ii)* pretransform and encode string-based data values to integer values to efficiently perform queries; *(iii)* use parallel query processing in case of complex queries. In case of many activities and a simple query structure, the parallelisation overhead might exceed the total execution time. Furthermore, *(iv)* be aware that querying event logs containing a big number of activities takes considerably longer than event logs with a smaller number of activities.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we provided a detailed analysis of database configuration parameters and their influence on query performance and storage requirements in the area of SQL-based process mining. While existing work on SQL-based process mining explains how to extract process constraints by SQL queries, it remained unclear how the underlying relational database needs to be configured to achieve the best query performance. In the work at hand we measured the influence of indexes, data types and parallel query processing on performance and storage requirements w.r.t. two real-life event logs. We discovered several significant findings that can serve as a configuration recommendation for process mining with SQL on relational databases. The research area of process mining based on SQL exhibits a number of different aspects that we want to examine in future work, including the efficient querying of event logs for data-aware declarative process models. Here, it might be necessary to integrate existing pre-processing and post-processing techniques in SQL-based declarative workflow mining.

ACKNOWLEDGMENTS

The work of Claudio Di Ciccio was funded by the Austrian Research Promotion Agency (FFG) under grant 861213 (CitySPIN). The work of Stefan Schönig was funded by the European Social Fund (ESF).

REFERENCES

[1] Elena Bellodi, Fabrizio Riguzzi, and Evelina Lamma. 2010. Probabilistic Declarative Process Mining. In *KSEM*. 292–303.
[2] Søren Debois, Thomas T. Hildebrandt, Paw Høvsgaard Laursen, and Kenneth Ry Ulrik. 2017. Declarative process mining for DCR graphs. In *SAC*. 759–764.
[3] Claudio Di Ciccio, Fabrizio Maria Maggi, Marco Montali, and Jan Mendling. 2015. Ensuring Model Consistency in Declarative Process Discovery. In *BPM*. 144–159.
[4] Claudio Di Ciccio, Fabrizio Maria Maggi, Marco Montali, and Jan Mendling. 2017. Resolving inconsistencies and redundancies in declarative process models. *Information Systems* 64 (2017), 425–446.
[5] Claudio Di Ciccio and Massimo Mecella. 2015. On the Discovery of Declarative Control Flows for Artful Processes. *ACM Trans. Management Inf. Syst.* 5, 4 (2015), 24:1–24:37.
[6] Taavi Kala, Fabrizio Maria Maggi, Claudio Di Ciccio, and Chiara Di Francescomarino. 2016. Apriori and Sequence Analysis for Discovering Declarative Process Models. In *EDOC*. 1–9.
[7] Fabrizio Maria Maggi, Jagadeesh Chandra Bose, and Wil van der Aalst. 2013. A Knowledge-Based Integrated Approach for Discovering and Repairing Declare Maps. In *CAiSE*. 433–448.
[8] Fabrizio Maria Maggi, Claudio Di Ciccio, Chiara Di Francescomarino, and Taavi Kala. 2018. Parallel algorithms for the automated discovery of declarative process models. *Information Systems* 74, Part 2 (2018), 136–152.
[9] Fabrizio Maria Maggi, Arjan Mooij, and Wil van der Aalst. 2011. User-Guided Discovery of Declarative Process Models. In *CIDM*. 192–199.
[10] Maja Pesic and Wil MP Van der Aalst. 2006. A declarative approach for flexible business processes management. In *BPM Workshops*. 169–180.
[11] Stefan Schönig, Cristina Cabanillas, Claudio Di Ciccio, Stefan Jablonski, and Jan Mendling. 2018. Mining team compositions for collaborative work in business processes. *Software and System Modeling* 17, 2 (2018), 675–693.
[12] Stefan Schönig, Cristina Cabanillas, Stefan Jablonski, and Jan Mendling. 2016. A framework for efficiently mining the organisational perspective of business processes. *Decision Support Systems* 89 (2016), 87–97.
[13] Stefan Schönig, Claudio Di Ciccio, Fabrizio Maria Maggi, and Jan Mendling. 2016. Discovery of Multi-perspective Declarative Process Models. In *ICSOC*. 87–103.
[14] Stefan Schönig, Andreas Rogge-Solti, Cristina Cabanillas, Stefan Jablonski, and Jan Mendling. 2016. Efficient and Customisable Declarative Process Mining with SQL. In *CAiSE*, Vol. 9694. 290–305.
[15] Boudewijn F. van Dongen and Shiva Shabani. 2015. Relational XES: Data Management for Process Mining. In *CAiSE Forum 2015*. 169–176.
[16] Michael Zeising, Stefan Schönig, and Stefan Jablonski. 2014. Towards a Common Platform for the Support of Routine and Agile Business Processes. In *Collaborative Computing: Networking, Applications and Worksharing*.