24th International Conference on Business Information Systems (BIS 2021)

Big Data

https://doi.org/10.52825/bis.v1i.66

Published: 02 July 2021

Database-less Extraction of Event Logs from Redo Logs

Dorina Bano[1], Tom Lichtenstein[1], Finn Klessascheck[1], and Mathias Weske[1]

[1]Hasso Plattner Institute, University of Potsdam, Germany

Abstract. Process mining is widely adopted in organizations to gain deep insights about running business processes. This can be achieved by applying different process mining techniques like discovery, conformance checking, and performance analysis. These techniques are applied on event logs, which need to be extracted from the organization's databases beforehand. This not only implies access to databases, but also detailed knowledge about the database schema, which is often not available. In many real-world scenarios, however, process execution data is available as redo logs. Such logs are used to bring a database into a consistent state in case of a system failure. This paper proposes a semi-automatic approach to extract an event log from redo logs alone. It does not require access to the database or knowledge of the database schema. The feasibility of the proposed approach is evaluated on two synthetic redo logs.

Keywords: Event Log, Redo Log, Database Schema, Log Extraction, Process Mining

1 Introduction

An event log is an intrinsic ingredient of process mining that is comprised of process execution data [1]. In traditional process mining, event logs are extracted from the databases of a given organization. Typical event log extraction approaches require extensive access to the database tables and detailed knowledge about the database schema [2]. If this information is not available, other sources of information have to be tapped to extract event logs, primarily redo logs. Redo logs are an attractive choice, since these logs not only hold values stored in the database, but also a temporal ordering of the modifications that were applied to these data.

Several Data Base Management Systems (DBMSs), like Oracle RDBMS [1], provide redo logs to store conducted data operations and use them to bring the database into a consistent state in case of system failure. Existing approaches propose solutions on how to derive event logs from redo logs. These approaches rely on information about the database and its schema [3]–[5]. In this paper we argue that it is possible to extract an event log by just considering redo logs as a single source of information.

Therefore, this paper proposes database-less extraction of event logs from redo logs. We are following a two-step semi-automatic approach where in the first step the database schema is automatically inferred from the redo log. The inferred schema is evaluated by a domain expert and it is used to extract an event log based on selected case notion. In the second step, the database schema is used to correlate the redo log events into the event log traces. The feasibility of our approach is tested on two synthetic redo logs.

[1]https://www.oracle.com/database/technologies/

The reminder of this paper is organized as follows. Section 2 briefly discusses the basic notions needed to understand the rest of the paper. The approach for extracting an event log from redo log is described in Section 3. The related work are briefly discussed in section 4. Consecutively, section 5 provides an evaluation of the proposed approach whiles section 6 concludes the paper.

2 Preliminaries

This section introduces the basic notions and concepts regarding the event log and redo log, which we refer to throughout this paper.

The starting point of any process mining techniques is an event log, which is defined as a ordered collection of events. Each event is represented by three mandatory attributes: the case identifier, which identifies the process execution instance; the activity name, which represents a well-defined step of the process execution; and, the timestamp, which represent the time occurrence of the event. In addition, an event log can store several optional attributes such as resources and organizations, showing the business unit where the process is executed. All events pertaining to the same case identifier establish the case.

One can make use of a plethora of event log extraction approaches to construct an event log around a specific case notion from a database of a given organization [2], [6]. Therefore, it is possible to extract several event logs, each pertaining to a different case notion, from the same source of information. To achieve that goal, the extraction procedure requires access to the database and deep knowledge of the schema. However, this access and knowledge cannot always be secured. Instead, usually another kind of log is more ready available—*Redo logs*.

A Redo log stores all changes of the database as they occur. Each entry in a redo log corresponds to an transaction executed by the database system. Database systems like Oracle RDBMS enable redo logs to store the historic view on what has happened in the system. In a real-life scenario these logs are used to restore the database to the consistent state in case of system failures.

Each transaction in the redo log is represented as an SQL statement (called redo entry) and consist of (1) the *operation* made upon a certain database table i.e., insert, delete, and update; (2) the affected *attributes* and the corresponding values, (3) the *row id* on which the statement must be applied, (4) the *timestamp* of the statement occurrence.

Some examples of the redo entries are illustrated in Listing 1, which shows several redo entries in a medical database system. Suppose that a patient is admitted to a hospital and afterwards diagnosed by a doctor. Each statement represents a redo entry. In the first redo entry two attribute value are inserted into the *Patients* table. While in the second second one an update operation is made upon the *Diagnoses* table. The third redo entry includes a delete operation over the *Admissions* table.

```
1 insert into "SYSTEM"."PATIENTS"("ID","GENDER") values ('86','M') AAAT
    ; 08-JAN-2021 12:46:15
2 update "SYSTEM"."DIAGNOSES" set "ID" = '584' where "ADMISSION_ID" = '
    101' and ICD_CODE='P599' and ROWID = 'BADR'; 08-JAN-2021 12:45:33
3 delete from "SYSTEM"."ADMISSIONS" where "ID" = '32' and "PATIENT_ID"
    = '34' and ROWID = 'SABD'; 08-JAN-2021 12:46:27
```

Listing 1. Redo log fragment: each statement corresponds to an transaction made upon the database and is called redo entry

In this paper we provide a method for extracting an event log from a redo log without the knowledge of the underlying database schema. Below a detailed explanation of each step is

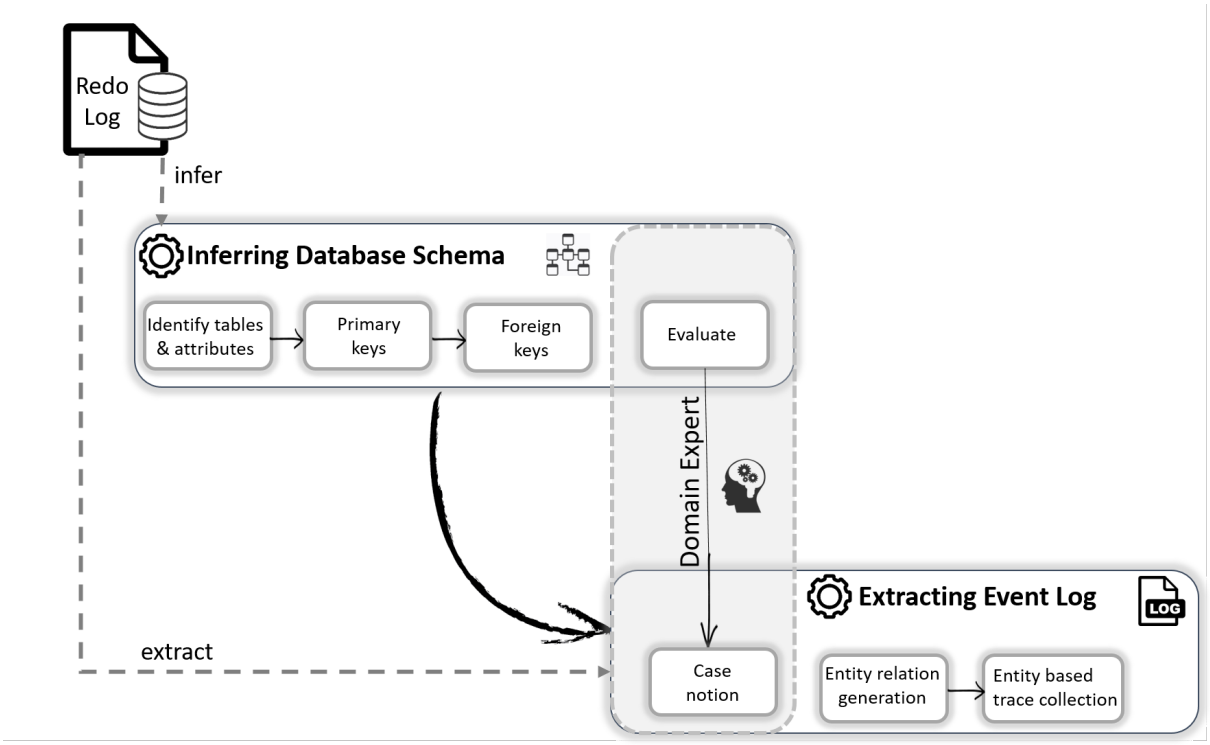

Figure 1. Overview of the approach followed to discover an event log from a redo log

given.

3 Extraction of event logs from redo logs

As mentioned in the previous section, the redo logs store all the changes of the database as they occur in the form of SQL statements. Our aim is to extract an event log by only considering the redo log as a single source of information as shown in Figure 1.

To correlate the redo entries into the event log traces a database schema is, nevertheless, needed. Therefore, the first step of our approach is to infer the database schema from a redo log without having access to the actual database. Afterwards, the domain experts comes into play to evaluate the inferred database schema and decide about the case notion by having in mind the goal of the process model. The last step of our approach is is to extract the event log from the redo log in conjunction with the database schema. The resulting event log is defined as a set of traces and each trace contains a set of events.

The assumption is that the redo log contains sufficient information to be able to extract the database schema and, in consequence, the event log. The bigger the redo log is and the more variation of the software system transactions it contains, the more accurate the inferred database schema becomes.

3.1 Inferring the database schema from redo logs

An important step that has to be taken before extracting an event log is to discover the database schema, which is used to correlate the data taken from the redo log into the event log traces. The database schema inferring process consists of identifying (1) the *database tables with their attributes*; (2) the *primary key* of each table; (3) the *foreign key* relations needed to determine the relation between tables.

Database schema tables and attributes detection To discover the database schema tables we look at each redo entry and filter out the table names. For each operation made upon the certain table in the redo entry a new database table is constructed. If the table name already exists in the database schema then we move forward to the next redo entry. Since each redo entry contains only one operation made upon one table then this step requires only one iteration over all redo log entries.

Once the database tables are constructed we have to identify for each table the affected attributes. Following the same idea, we go through all the redo entries and extract for each table the affected attribute names. If a new attribute is detected for a table, it is added to the corresponding table. Otherwise, the attribute is ignored as it is already in the schema.

The next step of the schema extraction is to define the relation between tables based on the attribute values involved in each redo entry. This implies the need to discover the primary keys and foreign keys. Since the redo logs do not contain explicit information about these types of keys we argue that this information can be extracted from the data pertaining to each attribute.

Primary key detection Primary keys are important constraints in the database indicating the attributes relations that hold in a database. By definition a primary key attribute contains only unique values [7]. This implies the need of checking the values of each attribute discovered from the previous step. If duplicate values appear for a certain attribute then this attribute is not a primary key candidate.

However, checking for unique values is necessary but not sufficient to detect the primary key. It might happen that a certain attribute contains unique values and can easily be misinterpreted as a primary key (e.g., an attribute which stores the value of the account balance). To overcome this issue, we further check if the attribute values appear in ascending order throughout the redo log. If this is not a case, the attribute is not considered as a primary key candidate even if its values are unique.

To increase the accuracy in the primary key identification step, we are also considering the attribute name suffix. In order to make the database more readable and easy to maintain, it is a common practice in reality for the primary keys to contain a suffix like: 'key', 'id', 'nr' [8].

Foreign key detection The foreign keys are used to identify the relations between the previously created tables. One difference between the primary keys and foreign keys is that we can have several foreign keys but only one primary key for each table. Another difference is that foreign keys require the existence of the primary key since they are used to reference a primary key.

To identify the foreign keys between the constructed tables and the predefined primary keys we rely on the *inclusion dependency* relation, which means that all the values of the referencing attribute must be contained in the referenced attribute. For example, if we have two attributes called X and Y respectively, each pertaining to two different tables in the database, we can say that all attribute values of foreign key Y must be present in the attribute values of the primary key X. This condition needs to be satisfied only on one direction, which implies that the primary key attribute might have additional values.

Nevertheless, the inclusion dependency is required but it does not suffice to detect the foreign keys. It might happen that there are, for example, 100 entries in the primary key attribute, 50 of which appear in another attribute of a different table. This means that this attribute is a candidate for foreign key. To increase the chances for discovering the correct foreign key in the same way as we did with the primary keys we will consider also the suffix attribute name.

Before using the inferred database schema as input for the event log extraction step we leave to the domain expert to review it.

3.2 Event log extraction

The event log extraction takes as input the redo log and the inferred database schema. It is the database schema that helps us to correlate the redo entries into the event log traces. The domain expert supports this transformation by picking the case notion while having in mind the desired process view. Specifically, the selected case notion denotes which database schema table will be used to determine the event log traces.

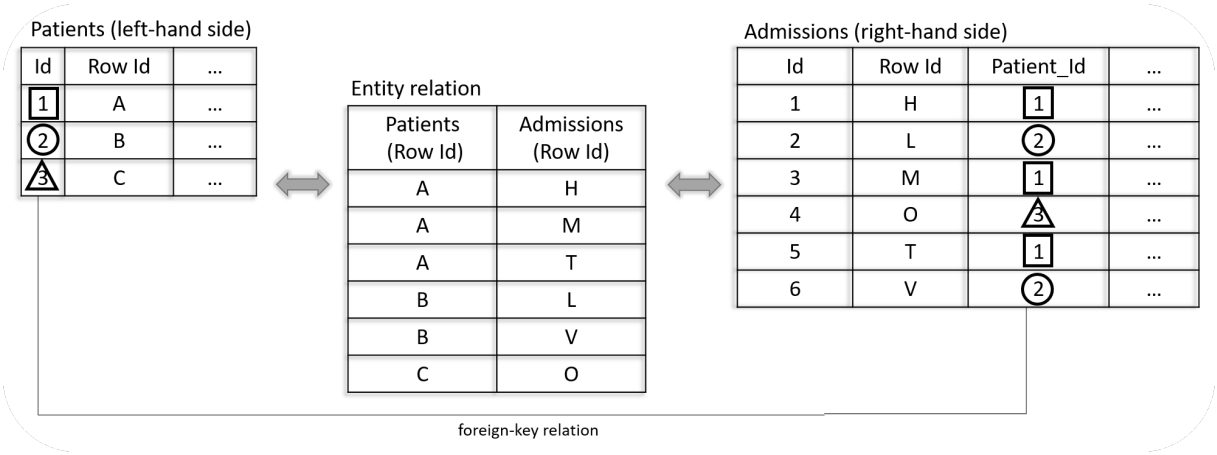

Figure 2. Example of an entity relation table creation. *Prescription* table is in relation with the *Pharmacy* table via the foreign key between *User Id* and *Id* attribute

Entity Relation Generation For each pair of tables in the database schema that are in relation via the discovered foreign keys the following method is applied: For each entity on the left-hand side of the referenced table it is checked whether one or more entities exist on the right-hand side of the depended table (i.e., the one that contains the foreign key attribute). If this is the case, the *Row Id* of the depended table together with the *Row Id* of the referenced table are added to an entity relation table.

This is done until all entities of the left-hand side table have been considered. The method is repeated for all tables that are in a foreign key relation outputting an entity relation table for each pair. This requires the *Row Id* to be unique. However, it might happen that several rows in the database might have the same row id. This occurs if a delete operation has happened on that row before. If a certain row is deleted from the database, its row id can be reassigned to another row via the insert operation. Therefore, a pre-processing step is needed to overcome this limitation.

Before constructing the entity relation table the redo log is parsed and whenever a redo entry that contains a delete operation occurs it's row id is tracked. If another redo entry is inserted and it occupies one of the tracked row id, then a unique suffix is appended to that row id. This is repeated as soon as the next delete statement for that row id is encountered. In this way, we make sure that each row id of the parsed redo log belongs exactly to one redo entry.

Figure 2 provides an example how the entity relation table can be created between two tables (called *Patients* and *Admissions*) that are in relation through the foreign key. Suppose that from the previously inferred database schema we get the information that each table has an primary attribute called *Id*. In addition, the *Patient_Id* attribute of the *Admissions* table (right-hand side) is assigned as a foreign key of the *Patients* table (left-hand side). For the entity with *Id* equal to *1* in the *Patients* table we will check all entities in the *Admissions* table that have the same value to the foreign key attribute (i.e., *Patient_Id*). As a result there are three entities that satisfy this conditions (highlighted with rectangle). In consequence, the corresponding *Row Id* of both tables are added to the entity relation table (first three entities of the entity relation table).

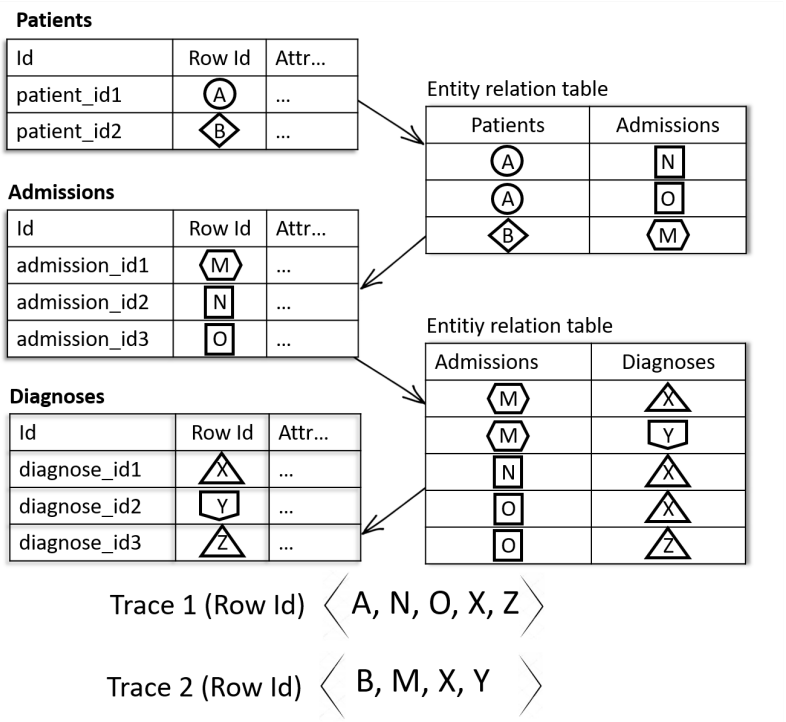

Figure 3. An example of event log extraction after applying the entity-based trace collection step on the inferred database schema. The Patients table is picked as root class/case notion by a domain expert

The same logic is followed for the entity with *Id* equal to 2 (highlighted with circle) and 3 (highlighted with triangle) in the *Patients* table. The resulted entity relation table has six relations in total. By default, there cannot be duplicates due to the uniqueness of the primary key. Once the relation is constructed for each pair of the tables that are in a foreign key relation, the next step is to define the event log traces.

Entity-based Trace Collection The goal of the entity-based trace collection step is to define the event log traces by considering the entity relation table and the redo entries. In process mining each trace is defined as as a collection of events. Each event is determined by a case id, activity name, timestamp and list of other optional attributes. To discover an event log trace we are considering for each row id of the root class (from now on let us call it RCID), picked by the domain expert, all row ids (from other tables) that are in relation with the RCID. This implies that for each RCID an event log trace is constructed.

To discover all row ids that are in relation with one RCID, the entity relation table comes into play. Starting from the the first RCID, we check the entity relation table for the entities that are in relation with this RCID. Let us call all entities that are in relation with the given RCID the *target entities relation*. Each target entity relation corresponds to one or more *Row Ids* of another table. Therefore the same step is followed to find all *Row Ids* that are in relation with the second table. In the same way we iterate through all tables for which a predefined entity relation table exists. For each RCID a trace is created as a list of all discovered Row Ids relating to it. Each event in this trace has a case id equal to the RCID entity. The event's activity name and timestamp are extracted from the redo log entries based on the respective Row Id event. Depending on the scope of the to-be-discovered business process model, several optional attributes can be defined via the attributes lists considered in each redo log entry. All the traces defined through this method constitute the event log.

Figure 3 illustrates an example of the entity-based trace collection based on the scenario

presented in Section 2. The patient was admitted to a hospital and afterwards diagnosed by the doctor. Suppose that the inferred schema for this scenario has three tables *Patients*, *Admissions* and *Diagnoses*. The Admissions table is in a foreign key relation with the Patients table, and Diagnoses is in a foreign key relation with Admissions. For each pair of tables the corresponding entity relation table is illustrated in Figure 3. Let us assume that the domain expert has selected the Patients table as a root class (case notion). For RCID equal to A the entity relation table is checked to identify the target entities relation. There are two target entities N and O in the entity relation table which at the same time defines the Row Id of the related table. In the same way we go further and check the related Row Id of N and O by considering the entity relation between Admissions and Diagnoses.

We iterate through all RCID of the root class and identify a list of Row Ids which are in relation with A and B. Therefore, from the example illustrated in Figure 3 two traces are discovered: $\langle A, N, O, X, Z \rangle$ and $\langle B, M, X, Y \rangle$. As the last step, for each *Row Id* in the predefined trace the redo log is queried to retrieve the activity name and time. The case id of the first trace is equal to the Row Id of RCID, which is equal to A in the Patients table while the case id of the second trace is likewise defined as B.

4 Related work

Discovering an event log from a redo log is recently subject to research work. Murillas et al. in [4] propose a three-step approach (scope, bind and classify) to extract an event log from the changes captured by the underlying database. The author assumes that the class model and event model, which capture the changes of the underlying database, are known upfront. In contrast, we are extracting an event log by considering only the redo log as a single source of information.

Another approach for extracting an event log from the redo log is presented in [5]. The event log extraction step requires three objects as input: the redo log, the data model and the trace id pattern. The latter is extracted from the database and it is used to determine the case notion needed to produce the event log traces. More specifically, the trace id pattern is used to find a common set of attributes between different classes while considering the primary key and foreign key relations. Instead, our approach assumes the lack of the database and such trace id pattern can be derived from the redo log entries.

In [3], the authors give an comparison between the traditional process mining and redo log process mining based on the event log extraction phase. They have applied both approaches on real-life data and come to the conclusion that the event log extracted from the redo log are richer in terms of number of events. In contrast to our approach the extraction of the event logs requires as input the redo log and the data model which is accessed via the connected Oracle database.

To the best of our knowledge, this is the first work on transforming database transactions into an event log without requiring live access to the database.

5 Evaluation

Our approach is evaluated based on two synthetic redo logs. The first one is extracted from the MIMC_III (Medical Information Mart for Intensive Care III) real-life dataset [9] which contains electronic health records (EHRs) related to patients admitted to the critical care unit (CCU) at the Beth Israel Deaconess Medical Centre (BIDMC), in Boston, USA. A script is written to simulate a redo log based on the provided MIMIC dataset. The approach presented in this paper is implemented as a Scala-based CLI tool and can be found on GitHub [2]. The Oracle

[2] https://github.com/fyndalf/redo-log-parser

Figure 4. Inferred database schema from the simulated MIMC redo log

LogMiner [3] is used to archive and export the redo logs.

The approach presented in Section 3.1 is applied to the simulated redo log and the inferred database schema is illustrated in Figure 4. After comparing it with the original schema, we come to the conclusion that they are similar.

As the next step, we chose two case notions: *Patients* and *Admissions*. As the last step of our approach the event log was extracted based on the selected case notion. The process model was then discovered from the extracted event log via the Inductive Visual Miner algorithm [10], which is a ProM plug-in [11]. The discovered process model is illustrated in Figure 5.

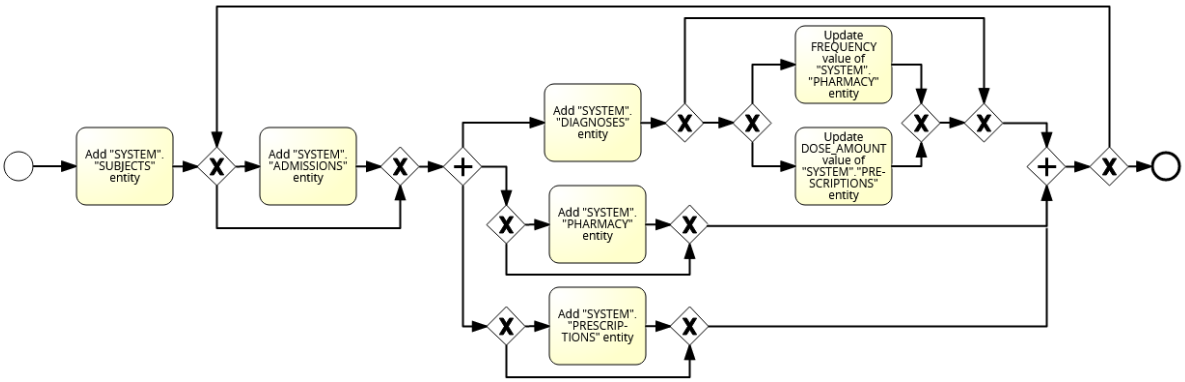

Figure 5. MIMC discovered process model for the Patient. The Inductive Visual Miner algorithm is applied with 70% threshold

From the discovered process we observe that the process starts with the insertion in the *Patient* table, which is happening because the *Patients* is selected as a root class. Afterwards, the insertion in the *Admissions* table can optionally happen. That is followed by the insertion in the *Diagnoses* table. In consequence the insertion of the *Prescriptions* or *Pharmacy* table can take place. The creation of all tables is followed by either updating the *Frequency* attribute in the *Pharmacy* table or updating the *DOSE_AMOUNT* attribute in the *Prescription* table which concludes the process.

Our approach is applied also to the *Ticket selling* dataset which used in [5]. This dataset

[3] https://docs.oracle.com/en/database/oracle/oracle-database/18/sutil/oracle-logminer-utility.html

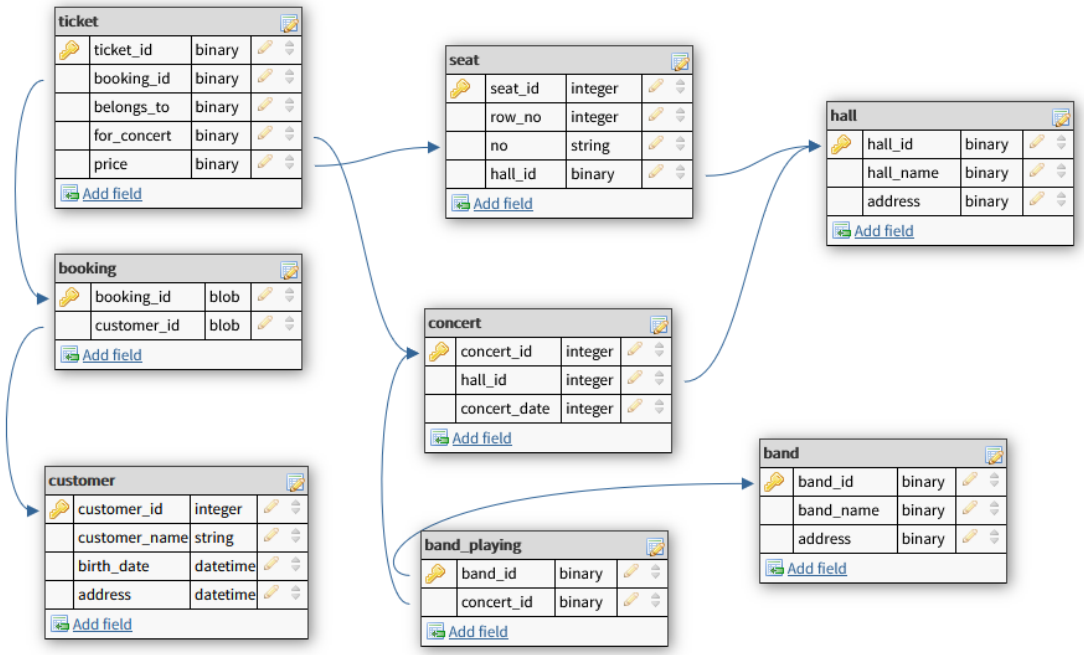

Figure 6. Inferred database schema from the Ticket selling redo log

contains events regarding a portal for selling concert tickets. The inferred database schema is shown in Figure 6 and it is compared with the original schema presented in [5]. There is only one extra relation inferred between *price attribute* in *Ticket table* and *no attribute* (seat number) in *Seat table* that is not manifested in the original schema. We checked the data and concluded that this seems to be an inconsistency in the original data: every ticket in the redo log has a price equal to 35 and there is one seat with the seat number equal to 35. We are assuming that the data are simulated and non-constant prices would not lead to this result.

6 Conclusion

In this paper we propose an approach to extract an event log from the redo log in the absence of a database. We show that the access to whole database and extensive knowledge about the database schema is not always necessary to extract an event log. To realize this, we propose a semi automatic two-step approach to output the desired event log. First, it is shown how to infer a database schema solely from the redo log. Then, the event log extraction is achieved by considering both the inferred database schema and the redo log. The assistance of the domain expert is required to select a case notion by having in mind the desired process model goal.

The feasibility of the approach is proven by developing a prototype which is applied on two synthetic redo logs. The inferred database schema it is proven to be similar to the original one proving the effectiveness of our approach. The discovered event log conforms to the XES [4] standard and can be used by the the process mining experts to apply different techniques like discovery, conformance checking and performance analysis.

As future work, we are aiming to further exploit redo logs as a rich source of information to enrich the discovered process model with additional information such as data objects. In addition we would like to further improve the performance and usability of our implementation tool.

[4]IEEE Task Force on Process Mining. XES Standard Definition. www.xes-standard.org

References

[1] W. M. P. van der Aalst, *Process Mining - Data Science in Action, Second Edition*. Springer, 2016, ISBN: 978-3-662-49850-7. DOI: 10.1007/978-3-662-49851-4. [Online]. Available: https://doi.org/10.1007/978-3-662-49851-4.

[2] S. Remy, L. Pufahl, J.-P. Sachs, E. P. Böttinger, and M. Weske, "Event log generation in a health system: A case study," in *Business Process Management - 18th International Conference, BPM 2020, Seville, Spain, September 13-18, 2020, Proceedings*, D. Fahland, C. Ghidini, J. Becker, and M. Dumas, Eds., ser. Lecture Notes in Computer Science, vol. 12168, Springer, 2020, pp. 505–522. DOI: 10.1007/978-3-030-58666-9_29. [Online]. Available: https://doi.org/10.1007/978-3-030-58666-9%5C_29.

[3] E. G. L. de Murillas, G. E. Hoogendoorn, and H. A. Reijers, "Redo log process mining in real life: Data challenges & opportunities," in *Business Process Management Workshops - BPM 2017 International Workshops, Barcelona, Spain, September 10-11, 2017, Revised Papers*, ser. Lecture Notes in Business Information Processing, vol. 308, Springer, 2017, pp. 573–587. DOI: 10.1007/978-3-319-74030-0_45. [Online]. Available: https://doi.org/10.1007/978-3-319-74030-0%5C_45.

[4] W. M. P. van der Aalst, "Extracting event data from databases to unleash process mining," in *BPM - Driving Innovation in a Digital World*, J. vom Brocke and T. Schmiedel, Eds., Springer, 2015, pp. 105–128. DOI: 10.1007/978-3-319-14430-6_8. [Online]. Available: https://doi.org/10.1007/978-3-319-14430-6%5C_8.

[5] E. González-López de Murillas, W. van der Aalst, and H. Reijers, "Process mining on databases: Unearthing historical data from redo logs," English, *Lecture Notes in Computer Science*, no. 9253, pp. 367–385, 2015, ISSN: 0302-9743. DOI: 10.1007/978-3-319-23063-4_25.

[6] K. Diba, K. Batoulis, M. Weidlich, and M. Weske, "Extraction, correlation, and abstraction of event data for process mining," *WIREs Data Mining and Knowledge Discovery*, vol. 10, no. 3, e1346, 2020. DOI: 10.1002/widm.1346. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1346. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1346.

[7] L. Jiang and F. Naumann, "Holistic primary key and foreign key detection," *J. Intell. Inf. Syst.*, vol. 54, no. 3, pp. 439–461, 2020. DOI: 10.1007/s10844-019-00562-z. [Online]. Available: https://doi.org/10.1007/s10844-019-00562-z.

[8] A. Rostin, O. Albrecht, J. Bauckmann, F. Naumann, and U. Leser, "A machine learning approach to foreign key discovery," 2009.

[9] A. E. Johnson, T. J. Pollard, L. Shen, L.-w. H. Lehman, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. A. Celi, and R. G. Mark, "Mimic-iii, a freely accessible critical care database," *Scientific data*, vol. 3, p. 160 035, 2016.

[10] S. J. J. Leemans, D. Fahland, and W. van der Aalst, "Process and deviation exploration with inductive visual miner," in *Proceedings of the BPM Demo Sessions Co-located with the 12th International Conference on Business Process Management (BPM)*, ser. CEUR Workshop Proceedings, vol. 1295, CEUR-WS.org, 2014, p. 46.

[11] B. F. van Dongen, A. K. A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and W. van der Aalst, "The ProM Framework: A New Era in Process Mining Tool Support," in *Applications and Theory of Petri Nets 2005, 26th International Conference, ICATPN*, ser. Lecture Notes in Computer Science, vol. 3536, Springer, 2005, pp. 444–454. [Online]. Available: https://doi.org/10.1007/11494744%5C_25.