MASTER

Database query log transformation for process mining application

Fakhri Afiff, F.

Award date:
2014

Link to publication
Database Query Log
Transformation for
Process Mining Application

Master Thesis

Fakhri Afiff

Supervisors:
Dr. D. Fahland(TU/e)
Prof. Dr. Martti Mantyla(Aalto University)

Assessment Committee Members:
Dr. M. Pechenizkiy(TU/e)
dr. I.T.P. Vanderfeesten(TU/e)

Final Version

Eindhoven, October 2014
Abstract

Process mining aim to provides multiple views on the same reality at different abstraction levels, instead of creating a single model of the process[1]. Such views should provide a purposeful abstraction of various behavior occurred during the process execution. Process mining techniques can provide models for the actual process behaviour, but mostly neglect the data flow perspective of the process execution. The data flow perspective is an important aspect of a process since a syntactically correct process sequence can still contain some errors due to incorrect data flow specification. However, there have been only limited discussion of the data flow perspective of a process in current literature. This motivates us to provide initial discussion on data flow discovery from database query log. In this thesis, we define several approach to discover the data flow, both from process perspective and artifact perspective. The presented approach is implemented and evaluated using an artificial data set. The evaluation shows some promising result, although there are still some limitations to it.

Keywords: Data Flow, Process Mining, Query Log, SQL, Event Log
List of Figures

1.1 Order to Shipment and Invoice to Cash Processes .......................... 2
1.2 Data Flow from Order Artifact .................................................. 3
1.3 Research Scope ................................................................. 4

2.1 Process Mining Overview[5] ...................................................... 8
2.2 Event Log Sample ................................................................. 8
2.3 General Event Log Structure[5] ............................................... 9
2.4 The Order artifact lifecycle .................................................... 10
2.5 Sample Query log from MySQL database ................................. 11
2.6 Overview of original XTract Approach .................................... 12

3.1 Overview of Database Query Log Transformation ....................... 16
3.2 Query Log with explicit case identifier .................................... 17

4.1 Discovered Data Flow For Order To Shipment ......................... 20
4.2 The artifacts from running example ......................................... 24

5.1 Prototype Architecture ......................................................... 28

6.1 Snippet from log table .......................................................... 32
6.2 Discovered data flow for Order to Shipment process ................. 34
6.3 Comparison of control flow and data flow for Order to Shipment process 35
6.4 Comparison of control flow and data flow for Invoice to Cash process 36
6.5 Control flow(left), data flow for each event(middle) and data flow(right) 36
6.6 Data flow of order to shipment compared to the full data flow .... 38
6.7 Discovered Data flow of Procure to Pay ................................. 38
6.8 Order to Shipment process map generated by filtering a full query log from various processes ................................................. 40
6.9 Discovered Data flow of order to shipment processes from query log from multiple processes without case identifier .................. 41
6.10 The artifact data flow of SaleQuotations artifact ................. 42
6.11 The artifact data flow of Requisition artifact .......................... 43

A.1 Database Schema .......................................................... 50
A.2 Procure to Pay Process Model ................................................. 51
A.3 Return to Supplier Process Model ........................................... 51
A.4 Order to Shipment Process Model ......................................... 52
A.5 Invoice to Cash Process Model .............................................. 52
A.6 Customer Return for Replacement Process Model ..................... 53

B.1 Sub Process Create Requisition Data Flow ............................ 54
B.2 Sub Process Create Quotation Data Flow ............................... 55
B.3 Sub Process Receive Quotation Data Flow ............................ 56
LIST OF FIGURES

B.4 Sub Process Create Purchase Order Data Flow ......................... 57
B.5 Sub Process Receive Purchase Order Data Flow ..................... 58
B.6 Sub Process Create Invoice Data Flow ............................... 59
B.7 Procure to Pay Data Flow using identifier Approach ................. 60
B.8 Order to Shipment Data Flow with Basic Approach .................. 61
B.9 Order to Shipment Data Flow using a combination of query pattern and event name 62
B.10 Order to Shipment Data Flow with Case Classification Approach .......... 63
B.11 The artifact data flow of Requisition artifact ..................... 64
B.12 The artifact data flow of Requisition artifact using a combination of query pattern and event name 65
List of Tables

3.1 Table Order in running example ........................................ 13
3.2 Query Pattern Example ..................................................... 14
3.3 Query Attributes Example .................................................. 14

4.1 The event log generated with the basic approach ....................... 20
4.2 The event log generated with the case classification approach .......... 22

6.1 Summary of query logs ..................................................... 32
6.2 Artifact Schema Definition ................................................ 33
6.3 Trace Identifier for each defined processes .............................. 37
Chapter 1

Introduction

This master thesis is carried out as part of EIT ICT Labs Master Program in Service Design and Engineering which is a joint collaboration between Aalto University and TU/e. This thesis has been carried out at the Architecture of Information Systems (AIS) group of the Mathematics and Computer Science department of TU/e.

This chapter starts with introducing the current state of business process analysis and potential usage of query log in section 1.1. It continues with the issue related with this query log utilization, which is formulated into the problem statement in section 1.2 and research scope in 1.3. The chapter concludes with the outline of this thesis in 1.4

1.1 Thesis Context

Process mining is an emerging research discipline that provides techniques that can be used to discover, monitor and improve real processes using event data. The goal is to extract knowledge from recorded data of an information system to obtain process-related information. Process mining can be distinguished into three different types. The first type is process discovery, where a process model is created using only the behavior observed in an event log. Process discovery is one of the most challenging process mining task [1]. The second type of process mining is that of conformance checking, where an existing process model is compared with an event log of the same process. This comparison shows where the execution of the process deviates from the process model. The third type is process enhancement, where a process model is extended or improved using information obtained from the event log. Combined, these process mining techniques can be used to check compliance, analyze bottlenecks, predict delays, and recommend actions to minimize the expected flow time.

In traditional process mining, event log is used as the input for analysis. The event log is a record of events that occurred in the process execution. In general, an event log contains the control flow of a single process; which means all high level events executed during a certain process execution. At its basic form, each event in the event log has 3 properties: a case ID, an activity and a timestamp. The case ID refer to which process instance the event belongs to. As its name suggest, activity refers to which activity the event related while the timestamp refers to the time when the event occurred.

Process mining aim to provide multiple views on the same reality at different abstraction levels, instead of creating a single model of the process[1]. Such views should provide a purposeful abstraction of various behavior occurred during the process execution. However, event log often do not contain information on the data perspective of a process execution. From data perspective, a process essentially is only a series of operation to various data objects stored in the database. On relational database, these operations take form as SQL query and most modern database management system have the capability to store every queries executed in a query log. Using this query log, we can discover the data flow in the system which generally is a translation of control...
flow on lower level. The data flow provides a precise description on what actually occurred in the information system and analyst can use this information to understand which data is affected by various events during the process and check whether the data is manipulated as intended. The data-flow can also provide description on various user behavior between events, for example which data is checked by user prior or after certain events. Furthermore, in a system where there are no event log presents, the data flow discovered from query log can be used to reverse engineer the process in order to rediscover the control flow.

The data flow from an information system can be seen from two different perspective. The first perspective is the data flow along the events occurred during a certain process execution, which is already previously described. The data flow from this perspective may include manipulation of multiple data object accessed during the process execution. The second perspective is along the events related to a single data object in the information system, which also often referred as artifact. An artifact are logical entities which represents specific data object manipulated during process execution. The sequence of operations performed to this artifact form a lifecycle, which may consist of events from multiple process. An artifact lifecycle generally start with the creation of an artifact instance (INSERT statement) and followed by other operations until the artifact reach its final state - if any.

Figure 1.1: Order to Shipment and Invoice to Cash Processes

To illustrate the difference between the two perspectives, we introduce a simple example which consists of two processes: Order to Shipment and Invoice to Cash. Order to Shipment process include activities related to customer sales process which begin when a customer requests a quotation or order goods to the moment the goods are shipped to customer. Invoice To Cash process is the continuation of Order to Shipment process which include activities related to invoicing and collecting payment from the customer. Figure 1.1 shows the process model from these processes and how each events in these processes accessed the database. The data flow for these process is shown by the dashed arrows between the corresponding process models and the database schema. For example, the data flow for each process is as follow:
CHAPTER 1. INTRODUCTION

The database shown in figure 1.1 contains two relevant artifacts: Order and Invoice. The Order artifact refers to a customer order whose lifecycle starts when the customer places an order and usually ends when it is paid for. The Invoice artifact refers to an invoice sent to request payment from a customer. Each instance of the Order artifact consists of Order and OrderItem tables, while the Invoice artifact only consists of the Invoice table. Each instance of the Invoice artifact is associated with one or more Order records from the same customer. The data flow for these artifacts represents their lifecycle and consists of all data operations conducted to the artifact. For example, figure 1.2 shows the data flow for the Order Artifact consisting of all data operations from Order to Shipment process plus a SELECT and UPDATE operation from Invoice to Cash process.

At the moment, there are no prior discussions on query log utilization for process mining analysis. Hence, this research can be considered as an exploratory research toward the utilization of query log in process analysis. However, this research is closely related to event log extraction from relational databases and we refer to some research conducted on this topic to develop our approach [3][5][7][10][8][9]. The main aim of this research is to explore the possibility of using query log for process mining analysis and identify the type of insights that can be acquired from it. In the process, we also identify various challenges in query log processing and propose an initial approach on query log utilization for process analysis.

1.2 Problem Description

Traditionally, query logs were used to analyze the performance of the database management system. There are no prior research on query log utilization for process analysis, but considering there are no explicit notions of activities and cases in the query log, it cannot be used as is for process mining analysis. Thus, a preprocessing step is required so that the query log can be used for process mining analysis. The main question is:

\textit{Given a database query log of a particular Information System, representing the execution history of one or several processes; how can we use it to discover the data flow in the system?}

The approach we took in this thesis is to transform query log into event log which can then be used on existing process mining tools. This transformation provides various challenges which are discussed in this thesis. The main challenge in transforming query log into event log is the fact that there are no notion of cases and activity in query log. At its basic form, the query log only contains a sequence of SQL queries and their timestamps. Classifying SQL queries into corresponding cases and activity is the main task in this thesis and depending on the form of query log, various cases are identified which affect the classification discussed in this thesis. Two different perspectives of data...
flow already discussed in previous section. Therefore the research question can be decomposed into two subquestions:

1. How to define a transformation from a given query log to an event log to discover the data flow of a certain process?
2. How to define a transformation from a given query log to an event log to discover the data flow of a certain artifact which form its lifecycle?

1.3 Research Scope

Given the research question, the goal of this thesis is to define an approach to transform query log into event log in order to enable query log utilization for process mining analysis. Since there are no prior research on the topic, we also would like to learn about the values of discovered data-flow, for example how the dataflow of a process compared to the control flow of the same process.

This thesis does not propose a new process mining technique and only focus in transformation query log to event log for a process or artifact. Subsequently, the classical process discovery algorithm can be used to obtain the control flow model of a process or a lifecycle model of artifact. The input for this procedure is a query log extracted from database which contain information about the data flow of a certain information system. The research scope is summarized in Figure 1.3.

For practical purposes, some assumptions were made to limit the scope of the project:

- The query log discussed in the thesis only contain basic SQL statement which include SELECT, UPDATE, INSERT and DELETE statement which only access a single table in the database. The rationale behind this is that this type of query generally is the direct representation of events of a normal process at data flow level.
- In addition to the query log, we also assumed that the data stored in relational database also available to use.
- The approaches developed is semi-automatic, which means additional human input would be required in the transformation process.

![Figure 1.3: Research Scope](image)

1.4 Thesis Outline

First, some preliminary concept related with this research is presented in chapter 2. Section 2.1 describe various concept related to process mining and event log. Section 2.2 describe some basic overview about database query log which followed by literature review on related research on section 2.3.
In chapter 3, we present in-depth analysis of SQL query and query log. Section 3.1 describe the problem analysis which include general characteristics of query log and some problems arise when it is used for process mining analysis, followed by an overview of the general transformation process in section 3.2. The detailed explanation about each transformation types is then presented in Chapter 4.

In chapter 5, we discuss about the implementation of the approaches defined in chapter 4. In order to validate the prototype, a thorough evaluation is conducted. This is explained in chapter 6. Finally, this thesis is concluded by Chapter 7 in which a summary of the thesis is given, and limitations of the mapping approach are discussed. We also include a list of future work to discuss the possibility to overcome the limitations and further improve the utilization of database query log for process mining analysis.
Chapter 2

Preliminaries

This chapter introduces preliminary concepts used throughout this thesis. Section 2.1 starts with a brief discussion on Process mining analysis and some framework which is used to analyze process information recorded in event log. Section 2.2 discuss the concept of event logs in general. Section 2.3 introduce the concept of database query log. Finally in section 2.4, some literature review on related research is discussed.

2.1 Process Mining

Process mining is a set of techniques to discover, monitor and improve real processes (i.e., not assumed processes) by extracting knowledge from event logs [11]. As can be seen in figure 2.1, process mining allow the discovery, analysis and extension of process models from which represent the actual process occurred in the real world. The process model contain information about the process flow, execution time, resource used and other relevant insights which would be very useful for business analyst to understand what actually occurred in real life, for example to check compliance, analyze bottleneck and detect deviating flows in the processes. However, traditional process mining assumed the existence of event log, which makes log extraction is out of the process mining scope. The problem of log extraction was explicitly added to Figure 2.1 to show its position in the classic process mining scope and help compare the traditional process mining scope to our research scope.

2.1.1 Control Flow Discovery

Control-flow discovery has been the cradle of process mining research and also one of the most challenging task in process mining. Control-flow discovery refer to automatically constructing a process model which describes the causal dependencies between activities. The basic idea of control-flow discovery is given an event log containing a set of traces, automatically construct a suitable process model describing the behavior seen in the log. The control flow perspective is described by one workflow process definition. A workflow process definition specifies which tasks need to be executed and in what order. A task is an atomic piece of work. Workflow process definitions are instantiated for a specific case. There are already various process discovery algorithm developed such as algorithm, fuzzy mining, heuristic mining, etc. These techniques will not be discussed here because the focus of this thesis is on the transformation process which enable the usage of query log as an input for these techniques. The output of the transformation is then used to construct a process map which represent the data flow of a process or artifact. The discovered model should be representative for the behavior seen in the event log; this is the so-called fitness requirement. In general, there is a tradeoff between the following four quality criteria of discovered process model [1]:

- Fitness: the discovered model should allow for the behavior seen in the event log.
CHAPTER 2. PRELIMINARIES

Figure 2.1: Process Mining Overview[5]

Figure 2.2: Event Log Sample

- Precision: the discovered model should not allow for behavior completely unrelated to what was seen in the event log.
- Generalization: the discovered model should generalize the example behavior seen in the event log.
- Simplicity: the discovered model should be as simple as possible.

2.1.2 Event Log

As previously mentioned, traditional process mining techniques assumed the existence of event log and use this event log as input. In general, an event log is a sequence of events recorded by information system. A sequence of events is called a trace or a process instance and is recorded from the tasks executed on a certain case instance going through the process. Figure 2.2 shows a fragment of event log for Order to Shipment process from our running example sample and its
A more generic structure of an event log can be seen in Figure 2.3 [5]. The right hand side part of the figure represents the traditional event log as presented by J. Buijs [2]. On the model level (or process definition level), a process specifies the activities which is executed in a certain order. A process can be instantiated to the case instances going through this process, also called process instances. The event types, instantiated from the activities, are performed on the case instances of the process which are recorded as events on the log level containing the data such as the time when the event is executed (i.e. timestamp), the resource who executed the event, and other attributes. An ordered list of events performed on a case instance is called a trace.

The left hand side of Figure 2.3 compares the definitions of event logs in a traditional mining context to the notions used in the artifact-centric context. An artifact represents the data object which is manipulated by one or many processes but in this context it can be seen as equivalent to the notion of a process definition, in which the event type definitions are specified. The artifact instances that share the same artifact type have similar event types. When extracting an event log based on a definition of an artifact type, each artifact instance is similar to a process instance (or a case) which results in a trace of events. For example, in our running example two artifact types are specified: Order and Invoice. An event log for order artifact contain events in which an order is manipulated. This events occurred in order to shipment and invoice to cash process as order is accessed by both process. An event log for invoice, however, only contain events from invoice to cash process since order to shipment process does not access invoice artifact.

There are several file formats used to store event log. Until recently, the de facto standard for storing and exchanging events logs was MXML (Mining eXtensible Markup Language). MXML emerged in 2003 and was later adopted by the process mining tool ProM. The current standard format for event logs is the XML based format XES, which is selected by IEEE Task Force on Process Mining. For further detailed explanation on XES file, we refer to the official website of the standard. Both J. Buijs [2] and H. Verbeek et al. [4] have discussed the difference between the MXML event log format and the XES event log format.

However, because of practicality issue, we decide to choose CSV (Comma Separated Value) as the format used to store the transformation output. The prototype implemented in this research is developed using Python and at the moment there are still no stable XES writer available for.
CHAPTER 2. PRELIMINARIES

Python. CSV file considered sufficient as it provide simplicity and flexibility required to store query event data. Some process mining tools also has the capability to process CSV file, while there are also stand-alone converter tool such as Nitro and ProMImport to convert CSV file into XES or MXML format.

2.1.3 Artifact Lifecycle

An artifact is a concrete, identifiable, self-describing chunk of information that can be used by a business person to actually run the business [6]. An Artifact represent a business object which is manipulated during a process execution and are described by both an information model and a non-trivial lifecycle. Artifact information is stored as records in the database table and each record is an instance of an artifact. However, an artifact instance can also contain multiple records in several tables. In this case, there is a main table which contains the instance information for the specific artifact: each instance can be identified by the primary key of the main table[7]. In our running example, the table Order is the main table for Order artifact, hence all instance of Order artifact can be identified with the value of OrderID. This is of course only under the assumption of such table exists for each artifact, which generally is the case. The artifact lifecycle describes all events occurred which directly related to the artifact. Figure 2.4 shows the lifecycle of Order artifact. The top model shows the artifact lifecycle from control flow perspective(events stored in event log), while the bottom model shows the artifact lifecycle on data flow perspective(SQL queries).

![Figure 2.4: The Order artifact lifecycle](image)

2.2 Query Logging

Most modern Database Management System equipped with a logging mechanism to record various database events, such as connect, commit, rollback command. The database log generally used by database administrator to provide an overview of the activities in the database. Database log generally used for transaction management (e.g. database recovery) and performance analysis (e.g. slow queries analysis). However, in this thesis we only refer to the database query log; which is a log containing a sequence of SQL query executed on the database. There are two ways in which we can acquire the query log. The first one is by extracting the queries from the standard transaction log created by the DBMS. However, as previously described, this log contain various database events, which means a filtering process is required to extract the query log. An example of this type of query log can be seen in figure 2.5.

Second is to use the query log created by the information system. It is quite common for BPM engine or other information system to have a query logging mechanism. This type of log generally more structured as it is stored in a table in the database, making it easy for us to extract specific information we are interested in. We used this type of log in this project because of its more structured nature; meaning no filtering process required. At this point, we also have no interest in other events in the database. Furthermore, this type of log might also contain various process related information(e.g. case ID, process type, the corresponding event in control flow), which makes it easier to analyze the data and also evaluate the constructed process model.

10 Database Query Log Transformation for Process Mining Application
2.3 Literature Review - Log Extraction

In recent years, the existence of event log sometimes become a hindrance in process mining analysis. This is especially evident in data-centric information system in which the data is stored in relational databases. In order to enable process mining analysis in such system, the event logs first has to be extracted from the relational data stored and many approaches have been developed for this purpose.

The traditional log extraction refer to the extraction event logs from data centric system based only on one notion of a case. The general step of these approach is as follow: define one notion of case and event type, then collect the events found in the data source that are associated with the case found and finally rewrite the data as an event log. These approaches extract one log for one process definition at a time, while assuming the process is isolated. Many issue has been found regarding these traditional log extraction approach. First, the data divergence and convergence issue. The data convergence is defined as the situation when one event is related to multiple cases, while the data divergence refer to the situation when one event is related to multiple cases. Furthermore, most traditional log extraction approaches are not general since it is specifically designed for the standard processes in SAP [3][10][8][9], hence extending these approaches to other systems or process will require more effort. J. Buijs was the first one to propose a general solution to support log extractions from data sources, however it still requires manual definition of the conversion process (i.e. mapping definition between the data source and event log) [2]

The limitation of traditional log extraction approach lead to the development of a more sophisticated approach called artifact centric log extraction approach. Instead focusing on process centric approach in traditional log extraction approach, the artifact centric approach describe the process as multiple collaborative artifact, each with its own artifact and life-cycle and interaction with each other. E. Nooijen [7] were the first to propose an automatic approach to identify artifacts from a given relational data and extract artifacts as event logs which then can be used as input for classical process mining techniques, referred as XTract approach. The overall method of this approach are shown in figure 2.6. The first step are the extraction of structural information (e.g. candidate keys and relationships between entities) from structured data (e.g. tables). Second, a subset of the dataset that is of interest for each specific artifact were identified. The idea in this artifact schema identification were to partition the full schema in a number of clusters (one for each artifact) and assign each table to one or more clusters. In addition a representative main table will be chosen for each cluster. This main table contains the instance information for the specific artifact: each artifact instance can be identified by the primary key of the main table. Third, a schema log mapping is automatically created for an artifact schema representing an artifact which is selected by users. Fourth, the mapping found in the previous step the used to automatically
generate event logs. As the last step, the XTract reused existing process discovery techniques to mine the life cycles of an artifact. We used this XTract approach as our starting point for artifact data flow discovery described in section 4.2. The XTract approach were later developed by X. Lu et al. [5] which enhanced the artifact discovery algorithm and add identification of artifact interactions.
Chapter 3

Database Query Log

This chapter provides an analysis of query log when used for process analysis. First, a description of the characteristic of SQL query and query log for process analysis are presented in section 3.1 and 3.2 respectively. In section 3.3, the general Transformation approach is discussed along with some defined assumptions in the transformation.

3.1 SQL Query Structure

The main challenge in processing the query log is the large number of variation in query. Unlike the event in event log which derived from a predefined activity, the flexibility in SQL query makes it possible to use various combination of custom attributes to perform a certain database operation. Consider the following records on Invoice table from our running example:

Assume that we are interested to fetch the first two tuple in the table, the following SQL queries perform an exact same action:

- SELECT * FROM Invoice WHERE OrderID=1
- SELECT * FROM Invoice WHERE InvoiceID=1 OR InvoiceID=2
- SELECT * FROM Invoice WHERE Amount > 700
- SELECT * FROM Invoice WHERE State='Finished' OR State='Pending'

Apart from the first two queries, we can replace OrderID with HandlerID and the queries still performing the same database operation. This variation in the query log makes it impossible to develop general approach which works on all query log since it is possible that queries from the same process instance does not have any similarity which can be used to correlate one another. The database used in our running example still relatively simple which makes the query variation is rather limited, but when the complexity of database schema goes higher, so does the query variation to perform a certain task.

In order to classify the queries, we define two components of SQL query:

1. Query Pattern
   - The query pattern is a set of basic SQL clauses and related attributes used in the query. We can see this pattern as an equivalence of activity in traditional event log. The query pattern consists of two part: the basic SQL clause and the SQL attributes. The SQL

<table>
<thead>
<tr>
<th>InvoiceID</th>
<th>OrderID</th>
<th>State</th>
<th>Amount</th>
<th>HandlerID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Finished</td>
<td>850</td>
<td>31231</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>Pending</td>
<td>750</td>
<td>19072</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>Invoice Sent</td>
<td>700</td>
<td>12983</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>Draft Created</td>
<td>450</td>
<td>12122</td>
</tr>
</tbody>
</table>

Table 3.1: Table Order in running example
command refer to the part of query which contains standard SQL clauses, which include
basic CRUD operation (SELECT, INSERT, UPDATE, DELETE) and other SQL clauses
such as WHERE, FROM, SET, AND, OR, ORDER BY. The SQL attributes refer to the
attributes used in the query and generally refer to the table and column name which is used
to limit the effects of queries. Consider the following query log excerpt from our running
example.

```
INSERT INTO order(OrderID, customer, state) VALUES (1, 'TU/e', 'created')
INSERT INTO OrderItem(OrderID, OrderIDD, Item, Price) VALUES (1, 1, 'Book', 10)
INSERT INTO OrderItem(OrderID, OrderIDD, Item, Price) VALUES (2, 1, 'Pen', 7)
INSERT INTO OrderItem(OrderID, OrderIDD, Item, Price) VALUES (3, 1, 'Pencil', 5)
SELECT * FROM order WHERE OrderID = 1
UPDATE order SET state = 'processed' WHERE OrderID = 1
SELECT * FROM order WHERE OrderID = 1
UPDATE order SET state = 'sent' WHERE OrderID = 1
```

The queries from the excerpt above has four distinct pattern which can be seen in table 3.2.

<table>
<thead>
<tr>
<th>Query Pattern</th>
<th>No. of Occurence</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT INTO order(OrderID, customer, state) VALUES (1, 'TU/e', 'created')</td>
<td>1</td>
</tr>
<tr>
<td>INSERT INTO OrderItem(OrderID, OrderIDD, Item, Price) VALUES (1, 1, 'Book', 10)</td>
<td>3</td>
</tr>
<tr>
<td>SELECT * FROM order WHERE OrderID = 1</td>
<td>2</td>
</tr>
<tr>
<td>UPDATE order SET state = 'processed' WHERE OrderID = 1</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 3.2: Query Pattern Example

2. Query Attributes
The query attributes refer to a set of attributes-value pair used in the WHERE clause of
the query. The query attributes are generally unique and can be used to determine the
uniqueness of the query. Two queries are assumed to be identical when both have the same
query pattern and attributes. Identical queries commonly generated by the same tasks and
a strong indication of a loop. Table 3.3 shows some query and its query attributes from
previous example.

<table>
<thead>
<tr>
<th>SQL Query</th>
<th>Query Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSERT INTO order(OrderID, customer, state) VALUES (1, 'TU/e', 'created')</td>
<td>OrderID=1, customer='TU/e', state='created'</td>
</tr>
<tr>
<td>INSERT INTO OrderItem(OrderID, OrderIDD, Item, Price) VALUES (1, 1, 'Book', 10)</td>
<td>OrderID=1, OrderIDD=1, Item='Book', Price=10</td>
</tr>
<tr>
<td>SELECT * FROM order WHERE OrderID = 1</td>
<td>OrderID=1</td>
</tr>
<tr>
<td>UPDATE order SET state = 'sent' WHERE OrderID = 1</td>
<td>OrderID=1</td>
</tr>
</tbody>
</table>

Table 3.3: Query Attributes Example

3.2 Query Log Characteristics
The introduction to event logs and query log described in previous chapter has shown that both
data structures differ from each other fundamentally. As discussed in 2.2, the query log we refer to
only contain basic SQL statement (which is a SELECT, INSERT, UPDATE and DELETE query)
from the Master database which means it contain a complete data flow of the information system.
In terms of process perspective, the data flow could be seen as translation of control flow at lower level. However, this data flow might not provide identical representation of the control flow in application layer. A single event of a process may generate one or more SQL queries. It is also possible for two distinct event to have identical SQL queries. If we also include the read operation (SELECT Query), we can also observe user behavior beyond the tasks which is not observble from event log. These makes the data flow generated from query log most likely to deviate from the process model generated from event log. Having the knowledge of both control flow and data flow of a process will be very helpful for conformance checking or process improvement. In case the control flow is not known (the system does not have an event log), the data flow can help domain expert to reverse engineer the process to discover its control flow.

Compared to an event log, a query log possess some distinguished characteristics:

1. **Query log contain information beyond task related event**
   The events stored in event log essentially is an execution of a certain predefined activity. This makes the insights gained from it limited to task based event, without much information on what happened between or during these tasks. Query log on the other hand contain data access event, which also includes user behavior between tasks. The observed behavior is in form of data lookup (SELECT Query) which executed when user. The insights about user behavior between task can be valuable input to improve the process, especially regarding the user experience aspect of it.

2. **There are no notion of activities and trace**
   One of the main differences between query log and event log is that an event log has a clear notion of activity and case. As explained in 2.1.3, each event in the event log has explicit definition on which activity was executed and to which process instance it belongs to. This makes correlating the event in the event log rather straightforward. Meanwhile, the basic query log only consist of two components: an SQL query and its timestamp. Most of these SQL statement are relatively unique and there will not be one attribute that can be used to correlate all queries of the same case to each other. The main task in transforming query log into event log will be to find the correlation between queries and this will be discussed in section 3.2.

3. **Queries with identical timestamp**
   In traditional event log, several events with identical timestamp rarely exists. In case it does exists, simultaneous process execution or inaccurate logging are most likely the reason behind it. It is quite unusual for a single process instance to have concurrent event with the same timestamp. On the other hand, in query log, the occurrence of multiple queries with identical timestamp is inevitable. The main reason is because a single event on application level may execute more than one SQL statement simultaneously.

4. **Identical queries**
   Generally, each unique event on application layer consist of one or many SQL queries with a unique set of attributes. Two queries with identical pattern and attributes are a strong indication of a loop in the process. However, there is also a possibility of two different events with fully or partially identical SQL queries. Unfortunately, without additional information, there are no way we can distinguish these two cases. This phenomenon might cause the process model constructed from query log to have a very different form than process model generated from event log and it can be measured. In this research, we assume identical queries only occurred when there is a loop in the process since the possibility of two event having identical query is relatively lower than a loop.

### 3.3 Database query log transformation

The transformation from query log into event log allows the discovery of data flow which can be seen from two different perspective: process data flow and artifact data flow.
1. **Process Data Flow**
   The traditional process mining aim to find the control flow of a process from event log. By using a query log, we can discover the process data flow which is a translation of control flow on the data level. Ideally, each event in the control flow have 1-to-1 or 1-to-many relation with an SQL query, resulting in some pattern similarity between the control flow and data flow.

2. **Artifact Data Flow**
   The artifact lifecycle is a sequence of operations performed to a particular artifact instance. Looking at control flow of a single process is not enough to get the full insights on artifact lifecycle since in most cases this lifecycle consists of events from multiple processes. Understanding artifact lifecycle can be useful to understand how the data evolve over time.

The transformation process for both process and data flow include 3 general steps as shown in figure 3.1.

![Figure 3.1: Overview of Database Query Log Transformation](image)

1. **Query Transformation** The first step are the extraction of various implicit information in the queries and store it in a more structured manner as a logical query object. We define the logical query object as an entity with several attributes :caseID, Query Pattern, Timestamp, Query attributes. The caseID of are only assigned in this step if there are explicit case identifier present in the query log, while the query attributes are extracted from the queries as discussed in section 3.1

   An important task in this step are to define an equivalence of activity of the queries. Unlike the events in event log which already have a clear notion of activity, the SQL statement in the query log is relatively unique to one another. We propose to use the query pattern previously described earlier in this chapter as an equivalence of activity. Using the query pattern, we can distinguish the queries and provide meaningful classification of the queries.

   The first step of the transformation is to classify each queries in the event log into a set of distinct activity. Unlike the events in event log which already have a clear notion of activity, the SQL statement in the query log is relatively unique to one another. However, the query pattern which we already discussed can be treated as the equivalence of activity. Using the query pattern, we can distinguish the queries and provide meaningful classification of the queries.

   Nevertheless this pattern based approach possess a potential pitfall. Suppose that there are two distinct event in the event log and these two event has identical pattern, the data flow will be misleading and looks very different with the control flow of the same process executions. For example, in order to shipment example, both prepare order and send order execute query with the same pattern: UPDATE order SET state WHERE OrderID; with state='processed' for prepare order and state='sent' for send order. This results in the two queries are treated as if it were the same activity, which makes the discovered data flow deviate from the control flow generated from event log. The solution for this problem is to manually include attribute value of the SET clause in certain query patterns or to refine the query pattern identification to discover such pattern. For
example, instead of `UPDATE order SET state WHERE OrderID`, we define two separate patterns:

1. `UPDATE order SET state='processed' WHERE OrderID`
2. `UPDATE order SET state='sent' WHERE OrderID`

2. Case Classification

This step refers to assignment of case identifier to a queries, making it possible to construct the traces by classifying queries with the same case identifier. A case identifier (CaseID) refers to a certain value which are used to classify related queries into the same trace. Depending on the perspective, case identifier can also be referred as process instance identifier or artifact instance identifier. A process instance identifier may present explicitly or implicitly. Explicit process instance identifier may be present when there are a separate logging mechanism inside the BPM system where queries are logged along with the information about the process type and its process instance identifier. An example of explicit process instance identifier are shown in figure 3.2. Implicit process identifier exists when there are a certain query attribute used which refers to a specific process instance. In this situation, we need to ensure its presence in every queries in the log by checking each queries either manually or automatically (e.g. using a simple program to check the presence of a certain attribute in the queries). In our running example, the `HandlerID` are an implicit process identifier as its value are unique for each process instance.

It is also possible that the queries doesn’t have any process instance identifier. In this situation, we argue that there are a set of attribute whose value is unique for each process instance referred as `trace keys`. The `trace keys` can be used to correlate queries from the same process instance and also define an artificial case identifier by choosing the one of the `trace keys` or combine the value of all `trace keys`. Generally, the table’s primary and foreign keys are the most likely candidate of `trace keys`. In our running example, an instance of order to shipment process is only a creation and manipulation of table `Order` and `OrderItem`. Using this knowledge, we can define the trace keys for order to shipment process consist of `OrderID` and `OrderIDD`. This approach will be elaborated further in section 4.1.2.

![Figure 3.2: Query Log with explicit case identifier](image)

An artifact instance identifier refer to a unique value which can be used to distinguish each instance of artifact of the same type. Generally, the primary key of artifact’s main table are the artifact instance identifier. In a situation where a certain artifact type only manipulated by a single process, we can also use the process instance identifier as artifact instance identifier. For example we can also use the value of `HandlerBy` from `invoice` table as artifact instance identifier because artifact invoice only accessed by a single process type: `invoice to cash`. More discussion on transformation for artifact data flow can be found in section 4.2.

3. Event Log Generation

After each queries already have a case identifier, the queries are now ready to be written as an event log. The output of this step are an event log, either in CSV, MXML or XES format. Regardless of the file format, we propose to store all of the query attributes in the generated event log. This information might be useful to correlate a certain phenomenon with the data stored in the database. For example, if we found that there are an unusual pattern in a hypothetical data flow, the attribute can be used to look whether there are a correlation between this pattern and the data stored in the database (e.g. it happened mostly to an order from certain customer). The attributes are stored with its table name used as...
prefix since it is possible that attributes from different table have a same name. For example OrderID from Order are stored as Order@OrderID, while OrderID from OrderItem are stored as OrderItem@OrderID. An example of the generated event log, stored as CSV file are as follows:

<table>
<thead>
<tr>
<th>caseID;time;pattern;table;Order@OrderID;...;OrderItem@HandlerID</th>
</tr>
</thead>
<tbody>
<tr>
<td>1;21:15:45 10/21/2014;INSERT INTO order(OrderID, customer, state);Order;1;...;</td>
</tr>
<tr>
<td>1;21:15:45 10/21/2014;INSERT INTO OrderItem(OrderID, OrderIDD, Item, Price);OrderItem;..;</td>
</tr>
<tr>
<td>1;21:15:45 10/21/2014;INSERT INTO OrderItem(OrderID, OrderIDD, Item, Price);OrderItem;..;</td>
</tr>
<tr>
<td>1;21:18:45 10/21/2014;SELECT * FROM order WHERE OrderID;Order;1;...;</td>
</tr>
<tr>
<td>1;21:18:55 10/21/2014;UPDATE order SET state WHERE OrderID;Order;1;...;</td>
</tr>
<tr>
<td>1;21:20:45 10/21/2014;SELECT * FROM order WHERE OrderID;Order</td>
</tr>
<tr>
<td>1;21:20:55 10/21/2014;UPDATE order SET state = 'sent' WHERE OrderID;Order;1;...;</td>
</tr>
</tbody>
</table>
Chapter 4

Process and Artifact Data Flow

This chapter provides more elaborate discussion on the approach proposed in this thesis. In section 4.1, the mapping for control-flow discovery is discussed along with problems and possible solutions related to it, whereas the mapping for artifact lifecycle discovery is discussed in 4.2.

4.1 Process Data Flow

The first transformation defined in this thesis are aimed for data flow discovery for a specific process. At database layer, an event in the application layer could be translated into one or more event on the database layer. The ideal data flow would have almost identical pattern with control flow as there should be 1-to-1 or 1-to-many relations between events in the control flow and queries in the data flow. We define 4 different approach based on different assumptions on the data. The assumptions used are as follows:

Assumption 1: The queries in the log already have case identifier of a process instance
Assumption 2: Query log only contains SQL operation from one process type

We start with a basic approach when both assumption hold in section 4.1, followed by dropping either one of the assumptions in section 4.2 and 4.3. Finally, we discuss an approach when both assumption are dropped in section 4.4.

4.1.1 Basic Approach

The basic approach are developed with two assumptions on the data to simplify the problem. This approach act as a starting point to develop other approach when the two assumptions are dropped later. The approach are quite straightforward:

Algorithm 1 Basic Process Data Flow

1: procedure BasicProcessDataFlow
2:   for Each lines in query the log do:
3:       Create a new query object Q
4:       Define the caseID for Q
5:       Generate event log

As previously described, the transformation process consist of 3 step: query transformation, case classification and event. The aforementioned assumptions makes it possible for us to skip the case classification since the queries already have a case identifier and it can be easily be classified into traces by using this case identifier. This means we only need to transform each queries into query object and generate the event log by using the data from all defined query objects. Consider the following query log excerpt from our running example.
CHAPTER 4. PROCESS AND ARTIFACT DATA FLOW

INSERT INTO order(OrderID, Customer, State, HandlerID) VALUES (1, 'TU/e', 'created', 8172)
INSERT INTO OrderItem(OrderIDD, OrderID, Item, Price, HandlerID) VALUES (1, 1, 'Book', 10, 8172)
SELECT * FROM order WHERE HandlerID = 8172
UPDATE order SET state='processed' WHERE OrderID=1 AND HandlerID = 8172
INSERT INTO order(OrderID, Customer, State, HandlerID) VALUES (2, 'Philips', 'created', 8173)
INSERT INTO OrderItem(OrderIDD, OrderID, Item, Price, HandlerID) VALUES (2, 2, 'Pen', 7, 8173)
INSERT INTO OrderItem(OrderIDD, OrderID, Item, Price, HandlerID) VALUES (3, 2, 'Pencil', 5, 8173)
SELECT * FROM order WHERE HandlerID=8172
UPDATE order SET state='sent' WHERE HandlerID=8172
SELECT * FROM order WHERE HandlerID=8173
UPDATE order SET state='processed' WHERE HandlerID=8173

Suppose that we already know that all queries above are generated from Order to Shipment process and we know that HandlerID refer to the process instance identifier of a process, we can use it as the case identifier of the queries and generate the event log shown in table 4.1. As discussed in chapter 3, in addition to table and query timestamp(not shown in the table), all query attributes are also stored in the event log. Using this event log, we can discover the data flow of the process as shown in figure 4.1

<table>
<thead>
<tr>
<th>CaseID</th>
<th>Query Pattern</th>
<th>OrderID</th>
<th>...</th>
<th>HandlerID</th>
</tr>
</thead>
<tbody>
<tr>
<td>8172</td>
<td>INSERT INTO order(OrderID, Customer, State, HandlerID)</td>
<td>1</td>
<td>...</td>
<td>8172</td>
</tr>
<tr>
<td>8172</td>
<td>INSERT INTO OrderItem(OrderIDD, OrderID, Item, Price, HandlerID)</td>
<td>1</td>
<td>...</td>
<td>8172</td>
</tr>
<tr>
<td>8172</td>
<td>SELECT * FROM order WHERE HandlerID</td>
<td>-</td>
<td>...</td>
<td>8172</td>
</tr>
<tr>
<td>8172</td>
<td>UPDATE order SET state WHERE OrderID AND HandlerID</td>
<td>1</td>
<td>...</td>
<td>8172</td>
</tr>
<tr>
<td>8173</td>
<td>INSERT INTO order(OrderID, Customer, State, HandlerID)</td>
<td>2</td>
<td>...</td>
<td>8173</td>
</tr>
<tr>
<td>8173</td>
<td>INSERT INTO OrderItem(OrderIDD, OrderID, Item, Price, HandlerID)</td>
<td>2</td>
<td>...</td>
<td>8173</td>
</tr>
<tr>
<td>8172</td>
<td>SELECT * FROM order WHERE HandlerID</td>
<td>-</td>
<td>...</td>
<td>8172</td>
</tr>
<tr>
<td>8172</td>
<td>UPDATE order SET state WHERE OrderID AND HandlerID</td>
<td>1</td>
<td>...</td>
<td>8172</td>
</tr>
<tr>
<td>8173</td>
<td>SELECT * FROM order WHERE HandlerID</td>
<td>-</td>
<td>...</td>
<td>8173</td>
</tr>
<tr>
<td>8173</td>
<td>UPDATE order SET state WHERE OrderID AND HandlerID</td>
<td>2</td>
<td>...</td>
<td>8173</td>
</tr>
</tbody>
</table>

Table 4.1: The event log generated with the basic approach

Figure 4.1: Discovered Data Flow For Order To Shipment

4.1.2 Handling Query Log without Case Identifier

In practice, it is more common for the queries not to have a case identifier. The general query log does not contain explicit process-related information(case identifier or process type of the executed queries). Furthermore, as previously mentioned, the queries can take multiple forms and it would be hard to guarantee the existence of this process instance identifier in every queries in the query log. Nevertheless, since assumption 2 still hold-every queries in the event log belongs to a single process type-it means we only need to find the correlation between queries from the same process instance.

One characteristics of a process is there are causal dependencies between events in a single process instance. This means that there are logical correlation between events in a single process instance which should also be reflected in database layer. On data perspective, this correlation is indicated by a specific set of table manipulated by a process instance. The trace keys are a set of attribute whose value is unique for each process instance. The trace keys can be defined with some knowledge on the database structure and process definition. By understanding what
the process actually do and having some knowledge on the database structure, the domain expert can define this trace identifiers. For example, if we look at our running example, HandlerID are a trace key for both process because its value is unique for each process instance. In order to shipment process, we can also argue that OrderID and OrderIDD is also trace keys because if we look at the process definition, there are 1-to-1 relation between an instance of order artifact and an instance of this process.

The method we propose consist of 2 steps: Trace Keys identification and query classification. After defining the trace keys type, we look at all queries in the log and identify a unique set of trace keys value for each trace. In addition, we can also extract the trace keys value from a SELECT query. Suppose that we have a query SELECT orderIDD from OrderItem WHERE OrderID=1, in addition to Order=1, we can also obtain the value of orderIDD by executing the query and use the value of obtained from the database. This, however, only applied if there are 1:1 or 1:n relation between the trace keys. Once we got a complete set of trace keys for each trace, we can classify the queries in the log into its corresponding trace by comparing the query attribute value and trace keys value. The approach are summarized as follows:

Algorithm 2 Process Data Flow - Case Classification

1: procedure ClassifiyerProcessDataFlow
2: for Each trace keys k do:
3:     for Each queries Q_ki which have k as its attribute do:
4:         TK_Q = all trace keys value in Q_ki
5:         if Q_ki is a select query with a trace key as its selector then
6:             Execute Q_ki and append the execution results to TK_Q
7:         if Exist Trace T with (k = k value in Q_ki) then
8:             Append all values of TK_Q Into T.keys
9:         else
10:             T = Create New Trace
11:             Append all values of TK_Q Into T.keys
12:     for Each discovered trace T do:
13:         Define the Artificial Process Instance Identifier
14:         Add all queries which has at least one of T.keys as its attribute
15:     Generate event log

To give a better understanding of the approach, consider the following query log from Order to Shipment process:

1-INSERT INTO order(OrderID,Customer,State,HandlerID) VALUES (1,'TU/e','created',8172)
2-INSERT INTO OrderItem(OrderIDD,OrderID,Item,Price,HandlerID) VALUES (1,1,'Book',10,8172)
3-SELECT * FROM order WHERE OrderID=1
4-UPDATE order SET state='processed' WHERE OrderID=1
5-INSERT INTO order(OrderID,Customer,State,HandlerID) VALUES (2,'Philips','created',8173)
6-INSERT INTO OrderItem(OrderIDD,OrderID,Item,Price,HandlerID) VALUES (2,2,'Pen',7,8173)
7-INSERT INTO OrderItem(OrderIDD,OrderID,Item,Price,HandlerID) VALUES (3,2,'Pencil',5,8173)
8-SELECT * FROM order WHERE HandlerID=8172
9-UPDATE order SET state='sent' WHERE HandlerID=8172
10-SELECT * FROM order WHERE OrderID=2
11-UPDATE order SET state='processed' WHERE OrderID=2

Suppose that we define OrderID, OrderIDD and HandlerID as the trace keys of the process. The first step are to discover each unique set of trace keys value. We begin by finding all unique value of OrderID in the log and we discover two traces; one with OrderID=1 and the other with OrderID=2 as trace keys. Then, we need to find whether the queries also use another trace keys as its attribute. For OrderID=1, we also discover HandlerID=8172 in the first query and OrderIDD=1 in the second query. For OrderID=2, we discover HandlerID=8173 in the fifth query, OrderIDD=2 in the sixth query and OrderIDD=3 in the seventh query. This result with the following trace keys for each traces: OrderID=1, OrderIDD=1 for the first trace and...
CHAPTER 4. PROCESS AND ARTIFACT DATA FLOW

HandlerID=8172; OrderID=2, OrderIDD=2, OrderIDD=3 and HandlerID=8173 for the second trace. Finally, we can generate an artificial caseID using these values, either by choosing one of the trace key value (e.g. caseID for Trace 1 = 1) or by concatenating all value (e.g. caseID for Trace 1 = 118172). The generated traces are shown in table 4.2 and the discovered data flow are already shown in figure 4.1.

<table>
<thead>
<tr>
<th>CaseID</th>
<th>Query Pattern</th>
<th>OrderID</th>
<th>...</th>
<th>HandlerID</th>
</tr>
</thead>
<tbody>
<tr>
<td>118172</td>
<td>INSERT INTO order(OrderID,Customer,State,HandlerID)</td>
<td>1</td>
<td></td>
<td>8172</td>
</tr>
<tr>
<td>118172</td>
<td>INSERT INTO OrderItem(OrderIDD,OrderID,Item,Price,HandlerID)</td>
<td>1</td>
<td></td>
<td>8172</td>
</tr>
<tr>
<td>118172</td>
<td>SELECT * FROM order WHERE OrderID</td>
<td>1</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>118172</td>
<td>UPDATE order SET state WHERE OrderID</td>
<td>1</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>2238173</td>
<td>INSERT INTO order(OrderID,Customer,State,HandlerID)</td>
<td>2</td>
<td></td>
<td>8173</td>
</tr>
<tr>
<td>2238173</td>
<td>INSERT INTO OrderItem(OrderIDD,OrderID,Item,Price,HandlerID)</td>
<td>2</td>
<td></td>
<td>8173</td>
</tr>
<tr>
<td>2238173</td>
<td>INSERT INTO OrderItem(OrderIDD,OrderID,Item,Price,HandlerID)</td>
<td>2</td>
<td></td>
<td>8173</td>
</tr>
<tr>
<td>118172</td>
<td>SELECT * FROM order WHERE HandlerID</td>
<td>-</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>118172</td>
<td>UPDATE order SET state WHERE HandlerID</td>
<td>-</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>228173</td>
<td>SELECT * FROM order WHERE OrderID</td>
<td>2</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>228173</td>
<td>UPDATE order SET state WHERE OrderID</td>
<td>2</td>
<td></td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.2: The event log generated with the case classification approach

4.1.3 Handling Query Log From Multiple Processes

In previous section, we already discuss an approach to handle query without case identifier. In this section, we will discuss about the case where assumption 2 is dropped while assumption 1 hold. In practice, it is quite likely that the query log contain queries from multiple processes, especially if the query log is taken from general query log. Since our goal is to discover the data flow of a certain process type, we need to filter out queries which is not generated by that process.

We propose to filter the log is to define a specific trace pattern (a set of query patterns) and use it as a reference trace to discover other similar traces. There are certainly variance between traces from a certain process, but we argue that there are a specific set of queries which always executed by each process instance of the same process type. Since we still assume that each queries already have case identifier, the trace can be easily constructed by connecting queries with the same case identifier. Then, we can compare each traces found in the log with reference trace and include only the traces whose similarity with the reference trace are above a certain threshold.

The similarity are measured by using percentage value of query pattern from the reference pattern found in the traces. For example, suppose we have two traces: T1 with 15 queries (12 unique pattern) and T2 with 18 queries (14 unique pattern). We define a reference trace consist of 10 query pattern. Out of 12 unique pattern in T1, 9 can be found in the reference trace, resulting in 90% similarity; while only 7 out of 14 pattern in T2 found in the reference trace making T2 only have 70% similarity with reference trace. If we use a threshold of 80%, only T1 should be included in the output because the similarity between T2 and reference trace are below the threshold. The log filtering approach is summarized as follows: It is still possible to filter the log even when the queries does not have case identifier, by filtering the queries individually using the pattern defined in base pattern. However, using the trace filtering yield a better result, as we can also include queries which is not in the pattern, as long as the trace it belongs to satisfy the pattern specified in base pattern. The base trace pattern mentioned previously can be prepared by a domain expert by executing a sample process and then extract the queries generated during the execution to be used as the base.
Algorithm 3 Process Data Flow - Log Filtering

1: procedure FilteringProcessDataFlow
2:  for Each Queries $Q$ do:
3:    if Exist Trace $T$ with caseID = caseID of $Q$ then
4:      Append $Q$ to $T$
5:    else
6:      $T = $ Create New Trace
7:      Append $Q$ to $T$
8:  for Each discovered trace $T$ do:
9:    Measure Similarity of $T$ with $TP$
10:   if Similarity $T$ with $TP_i \geq$ Threshold then
11:      Include $T$ in Query Log
12:     Generate event log

4.1.4 Handling Query Log From Multiple Processes Without Case Identifier

In previous sections, we already discuss the case when each of the assumptions were dropped one at a time. In this section, we will discuss an approach when both assumptions are dropped at the same time but we still assume the existence of trace keys described in section 3.3.

There are two steps required to handle this type of query log: filtering irrelevant queries (queries from other processes) and classifying the filtered queries into traces. Unlike the case discussed in 4.1.2 in which the filtering were done in trace level, the queries in this query log have to be filtered individually because the lack of case identifier in the queries. Individual query filtering can be done by referring to a set of reference patterns. The domain expert can run the process several times, covering all possible actions during the process, which generates a query log from that process. We can then extract unique patterns from the query log and use it to filter out the irrelevant queries from the log. The limitation of this approach is that queries from other process with the same pattern will most likely be included in the final output (false positive), which can affect the discovered data flow. After the filtering step, we can reuse the trace classification approach defined in section 4.1.1. The approach to handle query log from multiple processes without case identifier are summarized as follows:

Algorithm 4 Process Data Flow - Combination

1: procedure CombinationProcessDataFlow
2:  for Each Queries $Q$ in the log do:
3:    if $Q$ are in reference pattern $RP$ then
4:      Append $Q$ to Included Queries $IQ$
5:  for Each trace keys $k$ do:
6:    for Each queries $Q_{ki}$ in $IQ$ which have $k$ as its attribute do:
7:      if Exist Trace $T$ with $(k = k \text{ value in } Q_{ki})$ then
8:        if Exist Other Trace Keys $j$ in $Q_{ki}$ then
9:          Append $(j = j \text{ value in } Q_{ki})$ Into $T.keys$
10:     else
11:      $T = $ Create New Trace
12:      Append $(k = k \text{ value in } Q_{ki})$ Into $T.keys$
13:  for Each discovered trace $T$ do:
14:    Define the Artificial Process Instance Identifier
15:    Add all queries which has at least one of $T.keys$ as its attribute
16:    Generate event log
4.2 Artifact Data Flow

In previous section, we have discussed about the transformation for process data flow. In this section, we will discuss the transformation for artifact data flow; which also can be referred as artifact lifecycle. The first step in discovering artifact lifecycle is to extract the artifact schema from the database. This project will not propose a new artifact schema extraction approach, as we use manual identification of artifact schema. This manual identification of artifacts can be done by domain export by looking at the database schema used. The artifact schema described by Lu and Nooijen[5][7] has 3 properties: name, identifier and event types. However, since still do not have a definition of event types, we only need the domain expert to specify name and identifier for the artifact. Ideally, artifact identifier is sufficient to discover all events related to a particular artifact instance. However, in some cases there are also a possibility of queries without artifact identifier as its query identifier. For example, suppose that the invoice creation in Invoice to Cash process always created for all orders from a specific customer which is not yet invoiced. The following query then executed each time an invoice is created:

```
UPDATE Order SET state='invoiced' WHERE customer='Philips' and state != 'invoiced'
```

The query attributes customer and state are not an artifact identifier, which means without the records in the database, we could not correlate this query with corresponding artifact instance. Hence, we propose to include artifact records as the fourth attributes of an artifact. Figure 4.2 shows the artifact schema from our running example and some of the artifact instance exist in the database.

![Artifacts from running example](image)

Figure 4.2: The artifacts from running example

Generally, the approach to discover artifact lifecycle is very similar with the approach defined for control flow discovery. The main difference is that we need a full query log containing queries for every process executed in the system because it is very likely that a single artifact instance is manipulated by multiple processes. This means that all process related information (e.g. caseID, process type) are no longer relevant, as we are looking at artifact instance instead of process...
instance. The query log transformation for artifact lifecycle discovery can be summarized as follows:

Algorithm 5 Artifact Data Flow

1: procedure ArtifactDataFlow
2: for Each records of the main table do:
3:   Create an Artifact Instance A
4:   Fetch the artifact keys values for A
5:   Fetch all records belong to A from the database
6: for Each Artifact Instance A do:
7:   Find all queries Q using at least one of artifact key of A
8:   Include Q to the A event
9: for Queries Q in the list of unclassified queries do:
10: for Each Artifact Instance A do:
11:   if Query attribute of Q access one A’s record then
12:     Include Q to the A event
13: Create a trace for each artifact using queries in A event
14: Generate event log
Chapter 5

Implementation

This chapter discusses the implementation of the prototype of which the transformation was described in chapter 4. Section 5.1 discusses some important decision regarding the implementation. Section 5.2 discusses the file format used as the input and output of the prototype. Section 5.3 discusses the structure of the constructed script. The execution of the transformation is discussed in 5.4

5.1 Implementation decision

We have realized our approach by building a prototype to perform the transformation for both control flow discovery and artifact lifecycle discovery. The prototype is implemented using Python script. The main reason is because Python provide simple yet powerful platform. The goal of the prototype is to validate the idea and a Python script fit perfectly for this purpose, especially since no Graphical User Interface (GUI) is required for this purpose. Furthermore, Python interpreters are available for almost all operating system. The event log generated by the prototype then used as an input in Disco to conduct the process mining analysis. In order to make the modularity of the implementation clear, the prototype divided into 4 scripts. The correlation between each scripts can be found in figure 5.1.

1. **model.py**
   This script contain the class definition used in the transformation process. There are two class defined: SQLQuery and DataArtifact. SQLQuery class used to represent the queries from the log and used in both transformation. DataArtifact used to represent an artifact instance and used only in transformation for artifact lifecycle discovery.

2. **processDataFlow.py**
   This script contain the implementation of query log transformation for control flow discovery. This include: (1) basic transformation for query log with explicit identifier, (2) semi-automatic transformation for query log without explicit identifier, (3) automatic transformation for query log without explicit identifier and (4) log filtering for a specific process type.

3. **artifactDataFlow.py**
   This script contain the implementation of query log transformation for artifact lifecycle discovery.

4. **helper.py**
   This script contain other additional functions such as I/O function and various small parsing function.
5.2 Input and Output

This section describes the format of the file used as input and output of the prototype. There are several input for the prototype:

1. Query log
   The main input of the prototype which is stored as plain text file. Each line in the query log file contains information about one query. Each query instance consists of the SQL statement of the query, its timestamp, and additional process information such as case identifier (optional). To separate the value for each attribute, we use character "—" as separator. We choose "—" because this character is very unlikely to be present in the SQL statement.

2. Trace identifiers
   Trace identifiers are used as an additional input for the approach defined in section 4.1.1. The trace identifiers are stored as plain text file, with each line referring to a single identifier. The identifiers are defined in the following format: tableName@attributeName.

3. Trace pattern reference
   Trace pattern reference are used as additional input for log filtering approach defined in section 4.1.2. The trace pattern reference file is a plain text file which contains several query patterns, each stored in a single line.

4. Artifact schema
   The artifact schema are used as additional input for artifact data flow discovery. The artifact schema file contains the definition of related tables and identifiers for a specific artifact type. The file consists of four components: related tables, artifact key, artifact identifiers and table relations (if any). These components are separated by "—" and each item inside is separated by commas. The artifact key is an artifact identifier which is used for identification of artifact instance; generally it is the primary key of the main table from the artifact. The table relations define the relationship between tables, and since we only handle 1-to-many relationships, it shows the foreign key of the child tables. The following are the artifact schema for Order artifact from the running example:

```
Order, OrderItem
------
Order@OrderID
------
```

![Figure 5.1: Prototype Architecture](image)
The output of the prototype is an event log which then can be used as an input for various process mining tools. Each event in the event log have at least have the following properties: (1) caseID, (2) queryPattern, (3) tableAccessed, (4) timestamp and (5) query identifier value. The event log is stored in CSV format. CSV file provide simplicity required to store query event data and flexibility during the analysis. Moreover, there are still no stable version of XES or MXML writer in python. We believe CSV file is sufficient to store all information generated in the transformation process.
Chapter 6

Evaluation

In this chapter, we present the results of the experiments that were performed in order to evaluate the approach discussed in chapter 4. The chapter is structured as follows. Section 6.1 describes the setup of the experiments conducted. Section 6.2 presents the data used for the experiments. Section 6.3 discussed about the execution of the experiment. Section 6.4 discuss about the evaluation results. Finally, the chapter concluded with a discussion on gained insights and possible improvements to the approach in section 6.5.

6.1 Experiment Setup

In this section we describes the setup of various experiments conducted using the approach we already discussed in chapter 4. This evaluation were conducted with two main objective:

1. Analyze the discovered data flow
   The first objective are to discover relevant insights and analyze various emerging pattern from discovered data flow. For the process data flow, this is done by comparing the data flow and control flow of a process. For artifact data flow, we analyze the correlation between queries in the data flow with its corresponding process.

2. Evaluate the performance of developed approaches under different assumptions
   The second objective are to evaluate the performance of various discussed in chapter 4 and identify its limitation. The performance were measured by comparing the data flow generated by these approaches with the data flow from the basic approach(using the two assumptions described in chapter 4).

Ideally the evaluation should be conducted using a query log from a real process execution. However, since there are no available query log from a real process execution, we use an artificial query logs generated specifically for this evaluation. The artificial query log used in this evaluation arguably were very limited compared to a real query log, both in terms of size and complexity of the process. This means it is quite likely there are various phenomenon which couldn’t be found in the artificial, which in results also limit the observation from evaluation. However, we still believe this artificial query log were sufficient to verify most of the concept discussed in this thesis.

6.2 Experiment data

The artificial query logs for the experiments was generated from an implementation of several processes on Activiti process engine. Activiti is a light-weight workflow and Business Process Management (BPM) platform which provide simple, yet robust implementation of processes. The engine runs on Java application; in this case we use Apache Tomcat 7 server. We use the extended version of the engine which also support SQL queries using MySQL database. There are 2 database
CHAPTER 6. EVALUATION

<table>
<thead>
<tr>
<th>Query Log</th>
<th>No. of process instance</th>
<th>No. of Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order to Shipment</td>
<td>40</td>
<td>1208</td>
</tr>
<tr>
<td>Procure to Pay</td>
<td>31</td>
<td>426</td>
</tr>
<tr>
<td>Invoice to Cash</td>
<td>19</td>
<td>146</td>
</tr>
<tr>
<td>Overall Log</td>
<td>n/a</td>
<td>2503</td>
</tr>
</tbody>
</table>

Table 6.1: Summary of query logs

used by the system: Activiti database which is the standard database used by Activiti engine to store various information and a custom database called ACME used to store other custom data object.

There are 2 logging mechanism in the implementation: the general query log mechanism which is built-in logging mechanism from MySQL engine and a custom query log mechanism on the application level which log every queries on ACME database. The general MySQL query log arguably contain more data compared to the custom query log as it stored every database transaction occurred in the database. However, since we are only interested in the SQL queries executed during the process execution, we need another preprocessing step to extract this queries and leave other irrelevant data. On the other hand, the query log from the system’s custom logging mechanism stores the SQL queries in a structured manner, complete with various process related information for each queries(e.g. process type, process instance identifier). Based on this reason, we decide to use custom log instead of database log. The limitation of using custom query log is that it is relatively error prone, there are a (small) possibility that there were some unlogged queries; either intentionally or unintentionally. The executed processes is obtained from a student assignment from 21071-DBL information system course at TU Eindhoven. This assignment consist of several processes implementation in Activiti along with a custom database used by the system (the ACME database). There are 5 processes defined in the implementation: invoice to cash, order to shipment, procure to pay, return to supplier and customer returns for replacement. We execute these processes multiple times, covering all possible actions, in order to generate the event log. However, we decide only to use query log from the first 3 processes, as the latter two were considerably relatively a simpler process which won’t provide much additional value to the evaluation. The complete process model for each processes and the database schema used can be found in appendix A. We generate separate query logs for each of these 3 processes plus another query log which contains queries from all process executed in the system. A summary of the logs generated can be found in table 6.1. There are two versions of each logs: one with process identifier and event name correlated with each queries and another one only contain queries and its timestamp, which in total results in 8 query logs. Furthermore, we also identify two main artifacts from the system database which is used for evaluation. Table 6.2 shows the summary for these artifacts.
# CHAPTER 6. EVALUATION

## Attributes

<table>
<thead>
<tr>
<th>Table</th>
<th>SalesQuotation</th>
<th>Requisitions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>salequotations</td>
<td>requisitions</td>
</tr>
<tr>
<td></td>
<td>salequotationlines</td>
<td>requisitionlines</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ArtifactKeys</th>
<th>salequotations@sqoid</th>
<th>requisitions@reqid</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>salequotations@handledby</td>
<td>requisitions@handledby</td>
</tr>
<tr>
<td></td>
<td>salequotationlines@lineidd</td>
<td>requisitionlines@reqidd</td>
</tr>
<tr>
<td></td>
<td>salequotationlines@sqoid</td>
<td>requisitionlines@reqid</td>
</tr>
<tr>
<td></td>
<td>salequotationlines@handledby</td>
<td>requisitionlines@handledby</td>
</tr>
</tbody>
</table>

**Table 6.2: Artifact Schema Definition**

## 6.3 Experiment Results

In this section, we will discuss the result of various experiments conducted. Section 6.2.1 discusses the evaluation results for process data flow, while section 6.2.2 discusses the evaluation results for artifact data flow.

### 6.3.1 Process Data Flow

This section discusses the evaluation for various approaches in process data flow discovery. As discussed in chapter 4, there are two main assumptions used on the data:

1. **Assumption 1:** The queries in the log already have case identifier of a process instance
2. **Assumption 2:** Query log only contains SQL operation from one process type

The 3 processes are used for all the experiment. However, we found that we couldn’t use the log from Procure to Pay process for the basic approach and log filtering experiment because there are no consistent process identifier available stored in the log. This is caused by the implementation decision by the developer to divide the process into several sub-process with each sub process, making each instance of sub process have different process identifier even though conceptually they are from the same process instance.

#### Basic Approach

The first experiments were conducted for the basic approach in which the two assumptions hold. We aim to identify several patterns emerging from the discovered data flow and also analyze the correlation between data flow and control flow of a process. The query log with explicit process instance identifier from the Order to Shipment and Invoice to Cash were used in this experiment. We will mostly focus on the data flow from Order to Shipment process, as this data flow contain almost all observable phenomenon in the data flow.

Figure 6.2a shows the discovered control flow for Order to Shipment process; while figure 6.2b shows Disco’s performance view of the same data flow which can be used to identify simultaneous query execution. The number in each arcs on figure 6.2b shows mean execution interval between subsequent queries. Very small interval between queries (e.g. less than 1 second) are a strong indication that the queries are generated as a result of a single user action.

The first noticeable pattern are the back and forth arrow between

1. `SELECT sqoid FROM salequotations WHERE handledby` and
2. `SELECT COUNT(sqoid) FROM salequotationlines WHERE handledby` (marked with red circle). This is a rather unusual pattern which can be interpreted as a loop. However, when we look at the interval between query executions (figure 6.2b), there are average interval of 109ms for downward arrow and an average interval of 32.9sec for the upward arrow. Based on this phenomenon, we can conclude that there are two subsequent tasks:

   1. `SELECT sqoid FROM salequotations WHERE handledby`  
   2. `SELECT COUNT(sqoid) FROM salequotationlines WHERE handledby`

   `SELECT sqoid FROM salequotations WHERE handledby`
Another interesting observation are the relatively have a high frequency and self loop in query \texttt{UPDATE salequotation SET state WHERE squoteid} (marked with green circle). It is quite likely that there are several events in the control flow generate queries with this pattern. This argument were strengthened by relatively high mean interval in the loop(29.9 secs) while there are no loop found in the control flow. We can also argue that this phenomenon is a result of the weakness of our pattern based approach. We will discuss about possible improvement of our approach in the next chapter.

Figure 6.3 shows a comparison between the control flow(left) and simplified data flow(right) of Order to Shipment process. Removing the \texttt{SELECT} query makes the pattern similarity between control flow and data flow more obvious. Furthermore, we argue that the events in control flow always include some type
of data manipulation, making SELECT queries can be ignored for this purpose. In order to clarify the results, we also identify the correlation between tasks in the control flow and query patterns in the data flow. At this point, we only use the event name, query pattern, event descriptions and frequency of both events and queries to identify this correlations. We will use the correlation between query-event name available in the query log to validate our analysis later. The solid lane were used when there were a clear correlation between query and task, while the dashed shows an approximate correlations.

![Figure 6.3: Comparison of control flow and data flow for Order to Shipment process](image)

As shown in figure 6.3, there are 4 possible relation type between events in the control flow and the query patterns in the data flow. The 1:1 relation occurred when there a single event generate exactly one unique pattern in the data flow. The 1:n relation occurred when a single event generate several unique query pattern in the data flow. We can perceive this 1:1 and 1:n are the ideal relations between events and query pattern since each distinct events should be translated into different data manipulation. The n:1 relation which occurred when several events generate queries with the same pattern. This relation mainly due to the limitation of our pattern based approach already discussed previously. Finally, the n:m relation occurred when multiple events generate an identical set of query patterns.

As shown in figure 6.3 and 6.4, the difference between control flow(left) and data flow(right) were quite noticeable. For figure 6.3, the main difference are the branch at the beginning of the control flow("Prepare Sales Order" and "Prepare Quotation") did not appear in the data flow. This is a strong indication that both events execute queries with identical pattern.

In order to validate our observations, we construct another data flow from the same process execution, this time we use a combination of event name and query pattern. As shown in figure 6.5, the new data flow(middle) have more similar pattern with the control flow(left) because queries with similar pattern generated by different events were separated. With the exception of "Manage Order" event, each tasks in the control flow can be mapped into one or more query pattern. However, we found that there are no queries executed for "Manage Order" event. This most likely caused by an error in implementation, making the queries were not logged in the query log.

**Case classification approach**

The second experiments were conducted to evaluate the approach defined in section 4.1.2. In this experiment we use query logs in which each queries does not have a process instance identifier. As previously
described in Section 4.1.2, we need to manually define the trace identifiers for each processes and the following attributes were defined as the trace identifiers for each processes:

The trace identifiers were identified by referring to the process definition and the database schema in use. For example, since we understand that each instance of Procure to Pay process are essentially a series of manipulation on a single requisition object, we can use the primary and foreign key from requisitions and requisitionlines. Based on our observation on the queries in the log, we also learned that the handledby attribute were occasionally used in the queries instead of tables the primary and foreign key. The value of this attribute are unique for each process instance, which means it can also be used as trace keys. Figure 6.6 shows the discovered data flow(right) compared to the actual data flow(right)
CHAPTER 6. EVALUATION

<table>
<thead>
<tr>
<th>Process</th>
<th>Trace Identifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order To Shipment</td>
<td>salequotations@handledby</td>
</tr>
<tr>
<td></td>
<td>salequotations@squoteid</td>
</tr>
<tr>
<td></td>
<td>salequotationlines@handledby</td>
</tr>
<tr>
<td></td>
<td>salequotationlines@squoteid</td>
</tr>
<tr>
<td></td>
<td>salequotationlines@lineidd</td>
</tr>
<tr>
<td>Procure To Pay</td>
<td>requisitions@reqid</td>
</tr>
<tr>
<td></td>
<td>requisitions@handledby</td>
</tr>
<tr>
<td></td>
<td>requisitionlines@reqid</td>
</tr>
<tr>
<td></td>
<td>requisitionlines@reqidd</td>
</tr>
<tr>
<td></td>
<td>purchaseorders@poid</td>
</tr>
<tr>
<td></td>
<td>purchaseorders@handledby</td>
</tr>
<tr>
<td></td>
<td>invoices@poid</td>
</tr>
<tr>
<td></td>
<td>invoices@handledby</td>
</tr>
<tr>
<td></td>
<td>quotations@quoteid</td>
</tr>
<tr>
<td></td>
<td>quotations@handledby</td>
</tr>
<tr>
<td>Invoice To Cash</td>
<td>saleinvoices@handledby</td>
</tr>
<tr>
<td></td>
<td>saleinvoices@sinvoiceid</td>
</tr>
<tr>
<td></td>
<td>salequotations@handledby</td>
</tr>
<tr>
<td></td>
<td>salequotations@squoteid</td>
</tr>
<tr>
<td></td>
<td>salequotations@squoteid</td>
</tr>
<tr>
<td></td>
<td>salequotations@s invoiceid</td>
</tr>
</tbody>
</table>

Table 6.3: Trace Identifier for each defined processes

of Order to Shipment process. Although, we manage to identify all 40 traces correctly, we can see that there were several query patterns missing from the discovered data flow. The first two query pattern cannot be correlated because both queries doesn’t have any trace keys as its attribute. Similar with the first query, there are two other queries (SELECT stocks FROM products WHERE product AND attribute and SELECT products SET stock WHERE product AND attribute) which could not be classified in the trace because it doesn’t have any trace keys as its attribute.

As a matter of fact, even when the queries use one or more trace keys as its attribute, it is still possible for it to be unclassified. This is caused by the lack of queries which uses multiple trace keys as its attribute. The presence of this type of query is crucial since it act as a link between the trace keys to classify queries into a certain trace. Without a query of this type, it is almost impossible to classify queries which uses different trace keys into a trace. A good example of such case is the Procure to Pay process which consist of 6 sub-processes. As shown in figure 6.7, our approach failed to construct data flow for this process. In this case, there are no queries which can be used to find correlation between trace keys. For example, there are no queries in which reqid and reqidd used by a single query. As a reference, the data flow for each sub process can be found in Appendix B.

Filtering query log from multiple processes

In this experiment, we evaluate the query log filtering approach for control flow discovery defined in Section 4.1.2. The experiment performed by first executing the order to shipment process once, and use the query log generated from this execution as a reference point to filter another larger query log which contain queries from various process type.

Since each processes have a relatively linear flow, there are are only a small variance in the traces. By using 0.8 as a threshold we already discover a complete data flow for order to shipment process as found in figure 6.8. The reference trace used for Order to Shipment process can be found below:

```
SELECT risk FROM businesspartners
UPDATE businesspartners SET risk WHERE company
INSERT INTO salequotations(customer,handledby)
SELECT squoteid FROM salequotations WHERE handledby
SELECT COUNT(squoteid) FROM salequotationlines WHERE handledby
UPDATE salequotationlines SET subline WHERE handledby
INSERT INTO salequotationlines(squoteid,subline,product,attribute,price,quantity,handledby)
SELECT COUNT(subline) FROM salequotationlines WHERE handledby
UPDATE salequotationlines SET subline WHERE handledby
```

Database Query Log Transformation for Process Mining Application 37
CHAPTER 6. EVALUATION

Figure 6.6: Data flow of order to shipment compared to the full data flow

Figure 6.7: Discovered Data flow of Procure to Pay
Query log from multiple process without case identifier

In this experiment, we evaluate the combination of query log filtering and case classification defined in Section 4.1.3. Similar with previous experiment, we generate a reference pattern which are used to filter the log then used the trace keys defined in Table 6.3 to classify filtered queries into corresponding trace. We discover that there are no query patterns executed by multiple process type and this makes the output of this approach are identical with results in nearly identical data flow. For the Order to Shipment process, the results can be found in figure 6.9 which is almost identical with the data flow shown in figure 6.6, with only \texttt{SELECT lineidd FROM salequotationlines WHERE squoteid and state} missing from the discovered flow. This is due to the fact that this query pattern are not in the reference pattern we used. Note that in this case we use individual query pattern filtering instead of trace filtering described in previous section.

6.3.2 Artifact Data Flow

The fourth experiments were conducted to discover the artifact data flow which consist of queries from several processes and hopefully find some interesting insights regarding the artifact data flow in correlation with each processes. Based on the database schema of the system, 2 artifacts were identified from the database: Salequotations and Requisitions. Other artifacts in the system, such as Invoice and Quotations have relatively short and linear flow, hence will not be used in this experiment.

We use the artifact schema defined in section 6.2 and the experiment yield the data flow shown in figure 6.10 and 6.11. For SaleQuotations artifact, it is mostly accessed only by Order to Shipment process. The only new insights we can interpret from figure 6.10 is that Invoice to Cash were always preceded by Order to Shipment. The data flow for requisition artifact are more interesting. Since all sub-process of Procure to Payment artifact, we can see the correlation between these subprocess. In figure 6.11, we found that the Create Requisition sub-process always precede other 5 sub-processes. Following the Create Requisition are Create Quotation and Create Purchase Order. Looking at the figure, we can conclude Receive Quotation always preceded by Create Quotation, while Receive Purchase Order are always preceded by Create Purchase Order. Finally, the Create Invoice are the last part of the data flow in which it has to be preceded by Create Quotation and Receive Quotation process.

These findings from the artifact data flow partly prove our hypothesis that artifact data flow can be used as an indication to determine the relation between different process type. Although, due to the limitation on the data used in this evaluation, this need to be proved further using another dataset which contain artifact used by multiple process type.
CHAPTER 6. EVALUATION

Figure 6.8: Order to Shipment process map generated by filtering a full query log from various processes
Figure 6.9: Discovered Data flow of order to shipment process from query log from multiple process without case identifier
Figure 6.10: The artifact data flow of SaleQuotations artifact
Figure 6.11: The artifact data flow of Requisition artifact
6.4 Analysis

In previous sections, we already discussed various findings from the evaluation results. Overall, we argue
that there are a clear correlation between the data flow constructed from query log and the event flow of
a process given that the data flow are properly constructed. However, the experiments also shows that
the main challenge in constructing data flow from query log are the huge variation in the queries in the
log. The two assumption we used in the data might not be exist in real life, which makes this task even
more challenging. Hence, we argue that the query log processing should be treated on case by case basis.

Overall, we already propose a method for query log utilization to discover data flow of a process.
The proposed methods can be used successfully to discover the data flow and we already describe typical
correlation and patterns emerged between control flow and data flow. However, this result are achieved
only when the query log contain sufficient information for process analysis. A query log contain sufficient
information when it contain process related information(e.g. case identifier, process type) or the following
statement fulfilled:

1. Query from the same process instance can be correlated by the values exist in query attribute.
2. There are a set of trace keys, each with unique values, which can be used to classify the queries into
   traces.

We realized that the approach presented here is still far from perfect and there are still big room for
improvement. Some of the major weakness of our approach are as follows:

1. Dependency towards additional input from user
   In all defined methods, some additional user input(e.g. artifact schema, trace key, trace pattern)
   are required. These input need to be defined properly in order to get satisfying results. In some
cases(e.g. very complex process with hundreds of table in the database), identifying the proper
   input would be really huge task.

2. Limitation of the pattern based approach
   We already mention several cases in which our pattern based approach failed to distinguish queries
   generated from different events. A possible improvement for this approach are to include some
   attribute values in the pattern(e.g. attribute value in SET clause for UPDATE query). Another
   possible improvement are to group query with identical timestamp from a single trace into one
   event, which should result in a data flow with higher similarity to event flow because we would have
   a direct translation of events as SQL queries.

3. Limited to simple query
   The approaches we defined only handle the basic SQL query with a single related table and no
   JOIN operation. In practice, it is likely the queries will be more complex and more sophisticated
   approach would be required to handle such queries.

Specifically in terms of evaluation, we believe more validation are still required. This is especially due
to the fact that we only used artificial dataset for our evaluation, there are still a big question mark whether
the proposed approaches will also work for larger more complex dataset. The next step in the research
should be to focus on the validation of these approaches on query log from more complex processes. We
also have shown that the data flow could provide valuable addition in process analysis and hopefully there
are more discussion on this topic in the future. We believe that this thesis already served its purpose to
provide foundation for further research on this topic.
Chapter 7

Conclusions

Within this master thesis, we have analyzed the utilization of query log to discover the data flow during a process analysis. Before presenting our approach, we argue that the query log can provide additional insight during process analysis to validate the data operations performed by each events in control flow. In addition, the data flow could also be used to reverse engineer the process control flow when there are no event log present in the system.

The main contribution of this work are to provide initial ideas on how the database query log can be utilized for process analysis. We have discussed various problems in the utilization of query log and propose two point of views for the discovered data flow: the process data flow and artifact data flow. We defined several approaches to discover the data flow from these two perspective under different assumptions on the query log data.

An evaluation have been performed for all of the proposed approach using an artificial data set in which we discover the correlation between data flow and control flow of a process. Furthermore we also identify the limitation of the proposed approach and suggest possible improvement.

7.1 Limitation and Future Work

In this section, we discuss the limitations of the work presented in this thesis and suggest possible future work.

Further Evaluation Using Larger Dataset

The artificial dataset used during evaluation has proven to limit our findings in the discovered data flow, especially in artifact data flow discovery. This means a further evaluation using a larger query log from a more complex process would result in more insights and pattern discovered in the data flow. Further evaluation could also validate whether various hypothesis and assumptions defined are still hold on larger datasets.

Pattern Based Approach

One of the main limitations are related with our decision to use pattern based approach to transform query log into event log. In a situation where there are multiple task with identical query pattern, the pattern based approach are not suitable since the discovered data flow could give misleading insight or would be really hard to interpret. A potential improvement are to include the values of some query attributes if there are a constant set of value for it(e.g. enumeration type) which is identical for each process or artifact instance. For example, if we know that the value of attribute state are limited to draft, accepted, ready, final, then we can include its value in the query pattern. The identification of such situation would still be open for discussion.

Multiple Queries with the same Timestamp

A possible solution could be to cluster queries with identical timestamp from the same trace into a single event in the event log. Since several queries can be generated by a single event, clustering them would
provide more accurate representation of the data operation of each event in the control flow. This solution would also partly solve the limitation of pattern based approach discussed above since it is less likely that two different approach generate a set of SQL queries.

Automated Approach

The approaches proposed in this thesis are still semi-automatic since it requires users input for case classification, log filtering and artifact schema identification. For case classification problem, a possible improvement could be to apply machine learning technique to classify queries from a single process instance into the same trace. It would also be interesting to apply artifact schema identification techniques proposed by X. Lu [5] for artifact schema identification to discover the data flow from artifact perspective.
Bibliography


Database Query Log Transformation for Process Mining Application 47
Appendix A

Activiti Process Description

In this appendix, we provide the description and models for each processes used in the evaluation. All processes uses the database shown in figure A.1. Five process are defined

1. **Produce To Pay**
   Produce to pay refers to all the activities related to the purchase of goods from external suppliers. The process model for produce to pay process can be found in figure A.2. The implementation of this process are split into six sub processes: Create Requisition, Request Quotation, Receive Quotation, Create Purchase Order, Receive Order and Create Invoice.

2. **Return To Supplier**
   Return to supplier refers to the return of purchased goods back to the supplier. The process model for return to supplier process can be found in figure A.2

3. **Order To Shipment**
   Order To Shipment refers to all the activities related to handling the orders from customer. Order to Shipment process starts when a customer requests a quotation or orders goods to the moment the warehouse staff ships the merchandise. The process model for produce to pay process can be found in figure A.2

4. **Invoice To Cash**
   Invoice To Cash refers to all the activities related to the payment of customer orders. Invoice to Cash is a continuation of Order To Shipment process which starts by invoicing customers deliveries and closes it by receiving payments from buyers. The process model for produce to pay process can be found in figure A.2

5. **Customer Return For Replacement**
   Produce to pay refers to all the activities related to returning items back from customers for replacement. The process model for produce to pay process can be found in figure A.2
APPENDIX A. ACTIVITI PROCESS DESCRIPTION

Figure A.1: Database Schema
APPENDIX A. ACTIVITI PROCESS DESCRIPTION

Figure A.2: Procure to Pay Process Model

Figure A.3: Return to Supplier Process Model
APPENDIX A. ACTIVITI PROCESS DESCRIPTION

Figure A.4: Order to Shipment Process Model

Figure A.5: Invoice to Cash Process Model
Figure A.6: Customer Return for Replacement Process Model
Appendix B

Evaluation Results

In this appendix, we presents various data flow discovered during the evaluation phase.

B.1 Procure to Pay Data Flow

Figure B.1: Sub Process Create Requisition Data Flow
Figure B.2: Sub Process Create Quotation Data Flow
Figure B.3: Sub Process Receive Quotation Data Flow
Figure B.4: Sub Process Create Purchase Order Data Flow
Figure B.5: Sub Process Receive Purchase Order Data Flow

APPENDIX B. EVALUATION RESULTS

Database Query Log Transformation for Process Mining Application
Figure B.6: Sub Process Create Invoice Data Flow
Figure B.7: Procure to Pay Data Flow using Identifier Approach
B.2 Order to Shipment Data Flow

Figure B.8: Order to Shipment Data Flow with Basic Approach
Figure B.9: Order to Shipment Data Flow using a combination of query pattern and event name
Figure B.10: Order to Shipment Data Flow with Case Classification Approach
APPENDIX B. EVALUATION RESULTS

B.3 Artifact Data Flow

Figure B.11: The artifact data flow of Requisition artifact
APPENDIX B. EVALUATION RESULTS

Figure B.12: The artifact data flow of Requisition artifact using a combination of query pattern and event name

Database Query Log Transformation for Process Mining Application 65