# Extracting Data Manipulation Processes from SQL Execution Traces

Marco Mori(✉), Nesrine Noughi, and Anthony Cleve

Precise Research Center, University of Namur,
Rue Grandgagnage 21, 5000 Namur, Belgium
{marco.mori,nesrine.noughi,anthony.cleve}@unamur.be

**Abstract.** Modern data-intensive software systems manipulate an increasing amount of data in order to support users in various execution contexts. Maintaining and evolving activities of such systems rely on an accurate documentation of their behavior which is often missing or outdated. Unfortunately, standard program analysis techniques are not always suitable for extracting the behavior of data-intensive systems which rely on more and more dynamic data access mechanisms which mainly consist in run-time interactions with a database. This paper proposes a framework to extract behavioral models from data-intensive program executions. The framework makes use of dynamic analysis techniques to capture and analyze SQL execution traces. It applies clustering techniques to identify data manipulation functions from such traces. Process mining techniques are then used to synthesize behavioral models.

**Keywords:** Data-manipulation behavior recovery · Data-oriented process mining · Data-manipulation functions

## 1 Introduction

Data-intensive systems typically consists of a set of applications performing frequent and continuous interactions with a database. Maintaining and evolving data-intensive systems can be performed only after the system has been sufficiently understood, in terms of structure and behavior. In particular, it is necessary to recover missing documentation (models) about the data manipulation behavior of the applications, by analyzing their interactions with the database. In modern systems, such interactions usually rely on dynamic SQL, where automatically generated SQL queries are sent to the database server. In this context, our paper aims at answering the following research question: *To what extent can we extract the data-manipulation behavior of a data-intensive system starting from its traces of database access?*

M. Mori—beneficiary of an FSR Incoming Post-doctoral Fellowship of the *Académie universitaire 'Louvain'*, co-funded by the Marie Curie Actions of the European Commission.


S. Nurcan and E. Pimenidis (Eds.): CAiSE Forum 2014, LNBIP 204, pp. 85–101, 2015.
DOI: 10.1007/978-3-319-19270-3_6

The literature includes various static and dynamic program analysis techniques to extract behavioral models from traditional software systems. Existing *static* analysis techniques [1–5], analyzing program *source code*, typically fail in producing complete behavioral models in presence of dynamic SQL. They cannot capture the dynamic aspects of the program-database interactions, influenced by context-dependent factors, user inputs and results of preceding data accesses. Existing *dynamic* analysis techniques [6], analyzing program *executions*, have been designed for other purposes than data manipulation behavior extraction. Several authors have considered the analysis of SQL execution traces in support to data reverse engineering, service identification or performance monitoring [7–11]. Such techniques look very promising for recovering an approximation of data-intensive application behavior.

In this paper, we propose a framework to recover the data manipulation behavior of programs, starting from SQL execution traces. Our approach uses clustering to group the SQL queries that implement the same high-level data manipulation function, i.e., that are syntactically equal but with different input or output values. We then adopt classical process mining techniques [12] to recover data manipulation processes. Our approach operates at the level of a *feature*, i.e., a software functionality as it can be perceived by the user. A feature corresponds to a *process* enabling different instances, i.e., *traces*, each performing possibly different interactions with a database.

This paper is structured as follows. Section 2 presents the basic elements of our framework in terms of artifacts and components and how to integrate those elements to recover the data manipulation behavior of a data-intensive system. Section 3 presents a validation based on a tool integrated with current practice technologies and a set of experiments showing completeness and noise of mined processes depending on the input log coverage. Section 4 discusses related work and Sect. 5 ends the paper showing possible future directions.

**Motivating Example.** We consider an e-commerce web store for selling products in a world-wide area. The system provides a set of features requiring frequent and continuous interactions with the database by means of executing SQL statements. For instance, the feature for retrieving products (*view_products*) accesses information about categories, manufacturers and detailed product information. Which data are accessed at runtime depends on dynamic aspects of the system. For example, given that a certain feature instance retrieves the categories of products before accessing product information we can derive that it corresponds to a category-driven search. If a certain instance accesses manufacturer information before product information we analogously derive that it corresponds to a manufacturer-driven search. By capturing and mining the database interactions of multiple feature instances, it is possible to recover the actual data manipulation behavior of the feature, e.g., a process model with a variability point among two search criteria.

## 2 Data Manipulation Behavior Recovery

Our framework supports the extraction of the data manipulation behavior of programs by exploiting several artifacts (see Fig. 1). We assume the existence

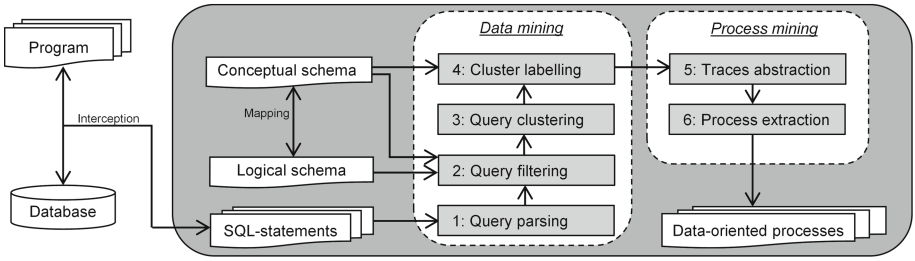



**Fig. 1.** Basics models: artifacts and components

of a *logical* and possibly of a *conceptual schema* with a mapping between them. The *conceptual schema* is a platform-independent specification of the application domain concepts, their attributes and relationships. The *logical schema* contains objects (tables, columns and foreign keys) implementing abstract concepts over which queries are defined. The conceptual schema and the mapping to the logical schema can be either available, or they can be obtained via database reverse engineering techniques [13,14]. *SQL statements* defined over the logical schema materialize the interactions occurring between multiple executions (traces) of a feature and the underlying *database*. Once the source code related to a feature has been identified [15], different techniques can capture SQL execution traces. Those techniques, compared in [8], range from using the DBMS log to sophisticated source code transformation. Among others, the approaches presented in [16,17] recover the link between SQL executions and source code locations through automated program instrumentation, while [18] makes use of tracing aspects to capture SQL execution traces without source code alteration. Once a sequence of queries is captured, it is necessary to identify the different traces, each corresponding to a feature instance. This problem has been tackled in the literature of specification mining by analyzing value-based dependencies of methods calls [19].

Our approach is independent from the adopted trace capturing techniques. For each feature, it requires as minimal input a set of execution traces, each trace consisting of a sequence of SQL queries. A *query parsing* component assignees to each query a set of data-oriented properties each describing its data-manipulation behavior. A *query filtering* component removes queries that do not refer to concepts and relationships of the input conceptual schema. A *query clustering* component clusters together queries that have the same set of properties thus implementing the same data-manipulation function. Consequently, a *cluster labeling* component identifies the signature representing the data-manipulation function (i.e., cluster) in terms of a label and a set of Input/Output (I/O) parameters. The *traces abstraction* component replaces traces of queries with the corresponding signatures and the *process extraction* component generates the *data-oriented process* corresponding to the input traces of a feature.

Noteworthy, our approach is applicable to any system for which a query interception phase is possible. It could, for instance, be applied to legacy Cobol

systems, Java systems with or without Object-Relational-Mapping technologies, or web applications written in PHP.

**Query Parsing (1).** We characterize SQL queries according to (1) the information they recover or modify and (2) the related selection criteria. To this end, for each query we record a set of data-oriented properties according to the query type. For a *select* query we record a property with the *select* clause while for *delete*, *update*, *replace* or *insert* queries we record a property with the name of the table. If the query is either *update*, *replace* or *insert* we also record a property with the *set* clause and all its attributes. Finally for all query types but the *insert* we add a property for the *where* clauses along with their attributes. By means of these properties we ignore the actual values taken as input and produced as output by each query. Figure 2 shows three SQL traces along with their corresponding properties. These queries are created starting from the logical schema represented in Fig. 3. Among others query $q_1$ is a select query over attribute *Password* of *Customer* table (property $p_1$) and it contains a *where* clause with an equality condition over *Id* attribute ($p_2$); query $q_2$ is a *select* query over attributes

Trace 1:

```
q1: SELECT Customer.Password FROM Customer WHERE Customer.Id = 'Mark27'; [p1,p2]
q2: SELECT Category.Id, Category.Image FROM Category; -> [p3]
q3: SELECT Product.Id, Product.Price FROM Product, PCategory WHERE Product.Id=PCategory.Product_Id AND
      PCategory.Category_Id='1'; -> [p4,p5,p6]
q4: SELECT PLang.Description FROM PLang, Language WHERE PLang.Language_Id=Language.Code AND PLang.Product_Id
      ='1A23' AND Language.Name='Italian'; -> [p7,p8,p9,p10]
q5: SELECT SpecialProduct.NewPrice FROM SpecialProduct,Product WHERE SpecialProduct.Product_Id=Product.Id
      AND Product.Id='1A23'; -> [p11,p12,p13]
q6: SELECT Manufacturer.Name FROM Manufacturer,Product WHERE Manufacturer.Id=Product.Manufacturer_Id AND
      Product.Id='1A23'; -> [p14,p15,p13]
q7: SELECT PLang.Description FROM PLang, Language WHERE PLang.Language_Id=Language.Code AND PLang.Product_Id
      ='1F32' AND Language.Name='Italian'; -> [p7,p8,p9,p10]
q8: SELECT SpecialProduct.NewPrice FROM SpecialProduct,Product WHERE SpecialProduct.Product_Id=Product.Id
      AND Product.Id='1F32'; -> [p11,p12,p13]
q9: SELECT Manufacturer.Name FROM Manufacturer,Product WHERE Manufacturer.Id=Product.Manufacturer_Id AND
      Product.Id='1F32'; -> [p14,p15,p13]
q10: INSERT INTO Log(IdEvent,Event,Date,Time) VALUES ('021','PrAcc1A23-1F32','2013-02-22','12:21:00'); -> [
      p16]
```

Trace 2:

```
q11: SELECT Customer.Password FROM Customer WHERE Customer.Id = 'JennyMa'; [p1,p2]
q12: SELECT Category.Id, Category.Image FROM Category; -> [p3]
q13: SELECT Product.Id, Product.Price FROM Product, PCategory WHERE Product.Id=PCategory.Product_Id AND
      PCategory.Category_Id='2'; -> [p4,p5,p6]
```

Trace 3:

```
q14: SELECT Customer.Password FROM Customer WHERE Customer.Id = 'DanWer'; [p1,p2]
q15: SELECT Manufacturer.Id, Manufacturer.Name FROM Manufacturer -> [p17]
q16: SELECT Product.Id, Product.Price FROM Product WHERE Product.Manufacturer_Id='AppleNamur01' -> [p4,p18]
q17: SELECT PLang.Description FROM PLang, Language WHERE PLang.Language_Id=Language.Code AND PLang.
      Product_Id='2D11' AND Language.Name='Italian'; -> [p7,p8,p9,p10]
q18: SELECT SpecialProduct.NewPrice FROM SpecialProduct,Product WHERE SpecialProduct.Product_Id=Product.Id
      AND Product.Id='2D11'; -> [p11,p12,p13]
q19: SELECT Manufacturer.Name FROM Manufacturer,Product WHERE Manufacturer.Id=Product.Manufacturer_Id AND
      Product.Id='2D11'; -> [p14,p15,p13]
q20: INSERT INTO Log(IdEvent,Event,Date,Time) VALUES ('022','PrAcc2D11','2013-02-28','14:00:03'); -> [p16]
```

SQL-statements properties:

```
p1="SELECT Customer.Password", p2="Customer.Id.EQ_VALUE", p3="SELECT Category.Id Category.Image",
p4="SELECT Product.Id Product.Price", p5="Product.Id=PCategory.Product_Id",
p6="PCategory.Category_Id.EQ_VALUE", p7="SELECT PLang.Description", p8="PLang.Language_Id=Language.Code",
p9="PLang.Product_Id.EQ_VALUE", p10="Language.Name.EQ_VALUE", p11="SELECT SpecialProduct.NewPrice",
p12="SpecialProduct.Product_Id=Product.Id", p13="Product.Id.EQ_VALUE", p14="SELECT Manufacturer.Name",
p15="Product.Manufacturer_Id=Manufacturer.Id",  p16="INSERT INTO Log",
p17="SELECT Manufacturer.Id Manufacturer.Name", p18="Product.Manufacturer_Id.EQ_VALUE"
```

**Fig. 2.** Web Store: Traces of SQL statements with data-oriented properties

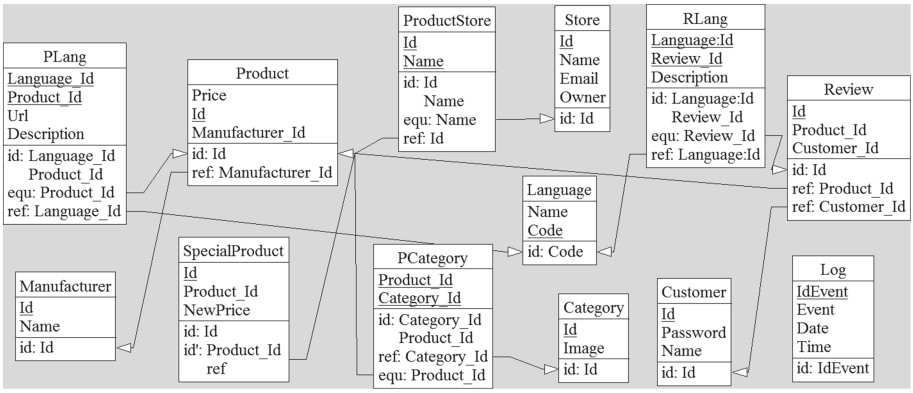



**Fig. 3.** Web-store case study: logical schema.

*Id* and *Image* of *Category* and it corresponds to property $p_3$; query $q_3$ is a *select* over attributes *Id* and *Price* of *Product* (property $p_4$), it contains two where clauses, i.e., a natural join between *Product.Id* and *PCategory.Product_Id* ($p_5$) and an equality condition over *PCategory.Category_Id* attribute ($p_6$).

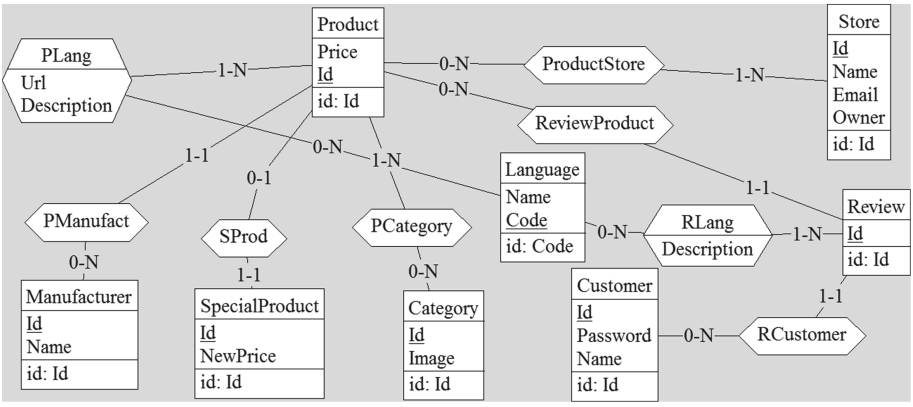



**Fig. 4.** Web-store case study: conceptual schema.

**Query Filtering (2).** We remove from the input traces the queries that do not express end-user concepts, i.e., the ones referring to database system tables or log tables appearing only in the logical schema. In our example we remove $q_{10}$ and $q_{20}$ accessing table *Log* without a counterpart in the conceptual schema (see Figs. 3 and 4).

**Query Clustering (3).** Starting from the traces of SQL statements we apply the Formal Concepts Analysis (FCA) [20] in order to cluster together queries

having the same data-oriented properties. FCA provides the definition for formal context $C = (O, A, R)$ where $O$ is the set of objects, $A$ is the set of attributes and $R \subseteq O \times A$ is the relation between objects and attributes. In our case objects are SQL queries while attributes are their data-oriented properties. For the formal context, a formal concept $c$ is defined as a pair $(O_i, A_i)$ where $O_i \subseteq O$ and $A_i \subseteq A$ and every object in $O_i$ has each attribute in $A_i$. In our case we assign each query to the concept with the same set of properties thus dividing queries in disjoint sets each performing different operations over the database. We report in Table 1 the clusters obtained from queries in Fig. 2. It is worth noticing that FCA is much more powerful than how we used it; indeed we did not consider objects (queries) having only subsets of equal properties. Nevertheless, we automatize the clustering of queries having the same set of data-oriented properties.

**Table 1.** Web Store: Clusters of SQL queries.

| C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 |
|---|---|---|---|---|---|---|---|
| $\{q_1, q_{11}, q_{14}\}$ | $\{q_2, q_{12}\}$ | $\{q_3, q_{13}\}$ | $\{q_4, q_7, q_{17}\}$ | $\{q_5, q_8, q_{18}\}$ | $\{q_6, q_9, q_{19}\}$ | $\{q_{15}\}$ | $\{q_{16}\}$ |
| $\{p_1, p_2\}$ | $\{p_3\}$ | $\{p_4, p_5, p_6\}$ | $\{p_7, p_8, p_9, p_{10}\}$ | $\{p_{11}, p_{12}, p_{13}\}$ | $\{p_{13}, p_{14}, p_{15}\}$ | $\{p_{17}\}$ | $\{p_4, p_{18}\}$ |

**Cluster Labeling (4).** We identify the data manipulation function implemented by each cluster in term of a label and a set of I/O parameters. First, labels are obtained by analyzing the fragment of conceptual schema which corresponds to the logical subschema accessed by the cluster queries. In case a conceptual schema is not available it is sufficient to reverse engineer the logical schema by simply adopting a data-modeling tool like DB-MAIN[1]; thus given that the logical schema contains meaningful names it is still possible to obtain significant labels. Second, I/O parameters are created based on the data-oriented properties belonging to each cluster.

For determining the labels we adopt the same naming convention proposed in [21] to associate conceptual level operations to SQL query code. We extract the portion of conceptual schema accessed by the queries of a cluster and we apply a different labeling strategy according to the query types. Concerning the query types *insert*, *replace*, *delete* and *update*, we create the label of the data-manipulation functions starting from the unique entity $E$ of the conceptual schema accessed by each of these query types, i.e., *InsertIntoE*, *ReplaceIntoE*, *DeleteFromE* and *UpdateE* respectively. In case of a *select* query we distinguish four cases according to the portion of the conceptual schema involved in the cluster queries (refer to Table 1 and Fig. 4 for the given examples):

a. One entity $E$. In this case we proposed two possible mapping names based on the presence of an equality condition over the primary key of $E$. If such condition is present, we map the cluster with the label *getEById*. Conversely, we simply map the cluster with the label *getAllE*. In our example we translate

[1] DB-MAIN official website, http://www.db-main.be.

cluster *C1* to $getCustomerById$ since queries in *C1* retrieve information contained within entity $Customer$ by taking as input its primary key. Concerning cluster *C2*, since its queries retrieve all tuples of entity $Category$ without considering any condition over its primary key, it translates to $getAllCategory$.

b. Two entities $E_1$, $E_2$ related by a many-to-many relationship $R$. In this case, the adopted label is $getAllE_1OfE_2ViaR$ providing that the queries give as result the attributes of all the instances of $E_1$ associated with a given instance of $E_2$ through $R$. Concerning our example we translate clusters *C3* to *getAllProductOfCategoryViaPCategory* since it extracts all the products of a certain category and we translate *C4* to *getAllLanguageOfProductViaPLang* provided that it extracts all language descriptions of a product.
c. Two entities $E_1$, $E_2$ related by a one-to-one relationship $R$. In this case, we map the cluster with the label $getE_1OfE_2ViaR$ provided that the queries retrieve the instance of $E_1$ associated to a certain input instance of $E_2$. In the web-store example, we map cluster *C5* to label *getSpecialProductOfProductViaSProd* in order to extract the occurrence of *SpecialProduct* related to a certain product via the one-to-one relationship $SProd$.
d. Two entities $E_1$, $E_2$ related by a one-to-many relationship $R$. In this case, we distinguish two cases. If the queries return the instance of $E_1$ that participates to the relationship $R$ with multiplicity N, we translate the query with the function $getE_1OfE_2ViaR$. Conversely, if the query returns the set of instances of $E_2$ that participate to $R$ with multiplicity 1 we translate the query with the label $getAllE_2OfE_1ViaR$. In our web-store example we translate cluster *C6* to *getManufacturerOfProductViaPManufact* since it retrieves the single occurrence of $Manufacturer$ participating to the relationship $PManufact$ with $Product$. We translate cluster *C8* to *getAllProductOfManufacturerViaPManufact* since it retrieves the multiple occurrences of $Product$ related to $Manufacturer$ via $PManufact$.

As far as I/O parameters are concerned, input parameters are the attributes involved in equality or inequality conditions that appear in the data-oriented properties of the queries, while output parameters are the set of attributes appearing within the *select* query property. Table 2 shows labels and I/O parameters for the clusters of the Web Store example.

It is worth noticing that in our approach it is enough to translate just one arbitrary query within the same cluster and to evaluate its input and output parameters; indeed all queries belonging to a cluster have the same set of properties and they consequently access the same portion of the conceptual schema. In defining signatures we have not considered the complete SQL grammar, e.g., we ignored *group by* operators that add more fined-grained information at the attribute level and we ignored the *where* clauses without value-based equality and inequality conditions. Nevertheless, we plan to adopt the latter for providing more detailed definitions of the data manipulation functions.

**Process Mining (5–6).** We generate a process starting from a set of SQL traces of a single feature. The *traces abstraction* phase replaces SQL traces with

**Table 2.** Web Store: Clusters with data manipulation functions and I/O parameters.

| Cluster | Input | Output |
|---|---|---|
| **C1**:*getCustomerById* | $\{Id\}$ | $\{Password\}$ |
| **C2***getAllCategory* | – | $\{Id, Image\}$ |
| **C3**:*getAllProductOfCategoryViaPCategory* | $\{Category\_Id\}$ | $\{Id, Price\}$ |
| **C4**:*getAllLanguageOfProductViaPLang* | $\{Product\_Id, Name\}$ | $\{Description\}$ |
| **C5**:*getSpecialProductOfProductViaSProd* | $\{Product.Id\}$ | $\{NewPrice\}$ |
| **C6**:*getManufacturerOfProductViaPManufact* | $\{Product.Id\}$ | $\{Name\}$ |
| **C7**:*getAllManufacturer* | – | $\{Id, Name\}$ |
| **C8**:*getAllProductOfManufacturerViaPManufact* | $\{Manufacturer\_Id\}$ | $\{Id, Price\}$ |

the corresponding traces of data manipulation functions. The *process extraction* phase exploits a process mining algorithm to extract the feature behavior as a sequence of function executions with sequential, parallel and choice operators.

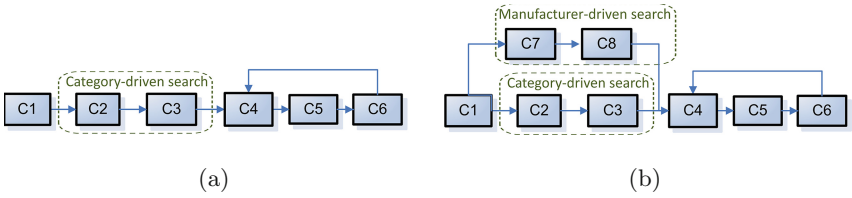
In the following we show how to recover the data manipulation behavior of the *view_products* web-store feature starting from the traces of data manipulation functions in Table 3 (obtained by replacing the queries in Fig. 2 with their corresponding cluster labels of Table 2).

**Table 3.** Web Store: Traces of data manipulation functions

| | |
|---|---|
| Trace 1 | *getCustomerById*(**C1**) - *getAllCategory*(**C2**) - *getAllProductOfCategoryViaPCategory*(**C3**) - |
| | *getAllLanguageOfProductViaPLang*(**C4**) - *getSpecialProductOfProductViaSProd*(**C5**) - |
| | *getManufacturerOfProductViaPManufact*(**C6**) - *getAllLanguageOfProductViaPLang*(**C4**) - |
| | *getSpecialProductOfProductViaSProd*(**C5**) - *getManufacturerOfProductViaPManufact*(**C6**) |
| Trace 2 | *getCustomerById*(**C1**) - *getAllCategory*(**C2**) - *getAllProductOfCategoryViaPCategory*(**C3**) |
| Trace 3 | *getCustomerById*(**C1**) - *getAllManufacturer*(**C7**) - *getAllProductOfManufacturerViaPManufact*(**C8**) - |
| | *getAllLanguageOfProductViaPLang*(**C4**) - *getSpecialProductOfProductViaSProd*(**C5**) - |
| | *getManufacturerOfProductViaPManufact*(**C6**) |

*Trace 1* gets customer information (*C1*), it performs a category-driven search of products by means of getting all the product categories (*C2*) and all the products of a certain selected category (*C3*). For each retrieved product, three functions are iterated: *C4* retrieves product description, *C5* extracts special

product information and *C6* extracts related manufacturer information. *Trace 2* is different from *Trace 1* because after function C3 no products are retrieved and the process ends. If we apply a mining algorithm to *Trace 1* and *2* we obtain a process (Fig. 5(a)) which performs consecutively functions *C1*, *C2* and *C3* before entering in the loop iterating *C4*, *C5*, and *C6*. The process ends after zero, one or more iterations of the loop. Let us now assume to include into the process *Trace 3* which is equal to *Trace 1* except that it searches products based on their manufacturer (functions *C7* and *C8*) instead of searching by category (*C2* and *C3*). If we mine the process model by considering as input all the traces (Fig. 5(b)), we end up with a new alternative branch: the customer can now perform either a manufacturer-driven search or a category-driven search.




(a) (b)

**Fig. 5.** Web Store: process mined with (a) *Trace 1* and *2* and (b) *Trace 1*, *2* and *3*.

## 3 Validation

We validated our approach to data-manipulation behavior recovery by means of *(i)* a tool integrated with current practice technologies and *(ii)* a set of experiments showing the sensitivity of our technique in producing correct processes depending on the traces log coverage.

### 3.1 Tool Support

The presented framework is implemented into an integrated tool which takes as input a set of SQL traces (each representing an instance of the same feature), the logical schema and optionally the conceptual schema and the conceptual-to-logical schema mapping. The tool relies on a set of implemented and third-party components. The former do not require any additional user inputs while the latter may require the intervention of the designer. Among the implemented components, a *SQL parser* extracts the data-oriented properties while a *filtering* component filters out the ones referring to concepts and relationships not in the input conceptual schema. A *clustering* component exploits the *colibri-Java* Formal Concept Analysis tool[2] to cluster queries according to their properties. A *labeling* component generates data manipulation functions (i.e., cluster signatures) while a *traces abstraction* component uses a Java library[3] to create standardized event logs.

[2] http://code.google.com/p/colibri-java/.
[3] http://www.xes-standard.org/openxes/start.

Starting from the event logs obtained with our implemented components, we rely on the de-facto standard process mining tool (*ProM* tool[4]) to mine data manipulation processes. *ProM* supports different process mining algorithms providing different trade-offs between completeness and noise [22,23] to be chosen according to specific application needs. In our case, we chose as miner algorithm the Integer Linear Programming (ILP) miner [24] since it is able to produce complete process models *(i.e., Petri Nets*[5] [25]*)* with a low level of noise[6]. Petri nets, which results from ILP miner, are well-semantically defined models enabling different types of analysis among which a precise comparison of different model instances. Once a process has been created, we exploit the ProM tool to export the Petri Net as a PMNL[7] file which can be given as input to a Petri Net editor tool, e.g., *WoPeD*[8] allowing reading and editing operations. ProM provides user-friendly graphical interfaces which support designers in easily loading standardized event logs, mining Petri Nets through ILP miner and exporting such models for editing purposes enabled by WoPeD.

### 3.2 Experiment Inputs

For our experiments we collected a set of database access traces from an e-restaurant web application developed by one of our bachelor students at our university. This data-intensive system provides different features each accessing a different portion of an underlying database to support the activities of taxi drivers, restaurant owners and clients. Clients consult menu information (*MenuInfo* feature), information about special offers (*DailySpecials* feature) and general information about restaurants (*RestaurantInfo* feature). They can reserve a table for a meal (*Reservation* feature) and they can issue orders of meals with two possible options, they either pick-up the meal at the restaurant or they

[4] http://www.promtools.org/.

[5] A Petri Net consists of a set of places, transitions, directed arcs and tokens. Transitions are represented with boxes and they indicate a certain event/task, places are represented with circles, directed arcs link together transitions and places in a bipartite manner, while tokens are represented as black dots which can move from one place to another trough a transition.

[6] In the literature of process mining two main metrics have been proposed to evaluate how good is an algorithm in mining models conforming to an input set of traces. The *fitness* measure expresses to what extent the model is able to produce the input traces (completeness), while the *appropriateness* measure expresses to what extent the model is able to represent the exact set of input traces (noise). Mining a process model which is both complete and without noise is not always possible unless we accept to obtain a model with too low level of abstraction (i.e., too specific) which does not help user readability. Therefore, between completeness and noise we give more importance to the first since we prefer to have a model which is able to reproduce all the input traces (fitness=1) even if we introduce a certain level of noise in it (appropriateness$<$ 1).

[7] http://www.pnml.org/ - (Petri Net Markup Language) is a proposal of an XML-based interchange format for Petri nets (de-facto standard).

[8] www.woped.org.

ask for a taxi service to deliver them the booked meal (*IssueOrders* feature). Restaurant owners prepare the meal to be delivered to clients (*RestaurantOrders* feature) while taxi drivers check delivery requests and they carry out the delivering process (*TaxiDelivery* feature). Among a wide set of implemented features we chose a subset of features whose data manipulation processes covered execution patterns of different nature, e.g., sequential execution, cycles and decision points. Since we played the role of the designer, we were able to select the most interesting features i.e., *DailySpecials*, *RestaurantInfo*, *MenuInfo*, *Reservation* and *IssueOrders*. Consequently we collected the corresponding sequences of SQL queries to give as input to the tool. The complete list of SQL statements grouped by feature with different traces and extracted data-manipulation functions are available at [26] along with conceptual and logical database schema accessible through DB-MAIN. We assume that for each feature, its input set of extracted traces corresponds to 100 % of coverage ratio of the process and it contains exactly 6 different traces.

### 3.3 Experiments

Starting from the inputs data we conducted a set of experiments aiming at answering the following research question: *What is the quality of the processes extracted through the integrated tool with a variable traces coverages, with respect to their correct versions identified by the designer?* In answering this research question, we organized the experiment in two different phases: the *start-up* phase creates for each feature the correct processes starting from its complete set of traces (traces coverage = 100 %) with the support of the integrated tool and the intervention of the designer; the *core* phase mines processes with a variable traces coverage ($\leq$ 100 %) and it evaluates the quality of such processes with respect to the correct ones previously identified with the support of ProM tool plug-ins.

**Start-Up.** For each feature we adopted the tool for creating the data-manipulation processes with the complete set of traces. Consequently, since these models could either contain noise or they could be not complete, we asked the designer to derive a correct version from it. Designers have a deep knowledge and understanding of the processes and they can easily assess if a certain process is correct or not. In our case, as designers, we adopted the WoPeD tool for visualizing the processes as Petri Nets and to perform the possible required modifications, i.e., addition/deletion of places and arcs. For instance, Fig. 6(a) shows the feature *DailySpecial* mined with our technique while Fig. 6(b) shows the version of the same feature as it has been corrected by the designer. The correct version of this feature mainly consists of retrieving a certain set of restaurants ($A$), checking if they provide special dishes ($B$) and for each of those dishes it retrieves the corresponding information ($C$) along with the category to which the dish belongs ($D$). The mined process contains all valid traces in the corrected process but, as a consequence of the noise introduced by the mining algorithm, it enables more traces than the correct one, e.g., the traces that retrieve a certain

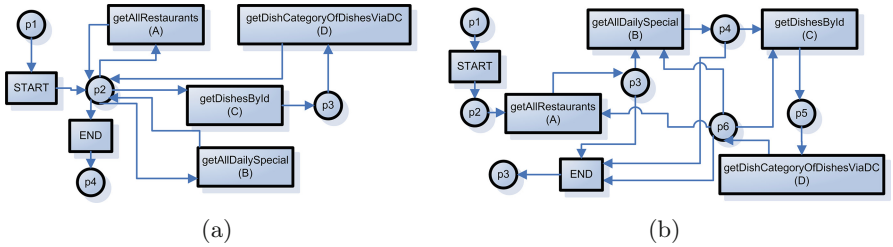


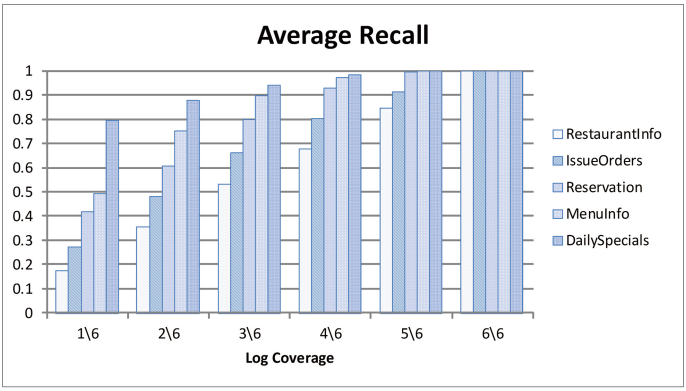

**Fig. 6.** *DailySpecial*: mined process (a) and process corrected by the designer (b).

dish ($C$) without first retrieving the special dishes ($B$). These traces are not considered admissible by the designer, who modifies the links among transitions to disable the former traces and forcing the process to perform $C$ only after $B$.

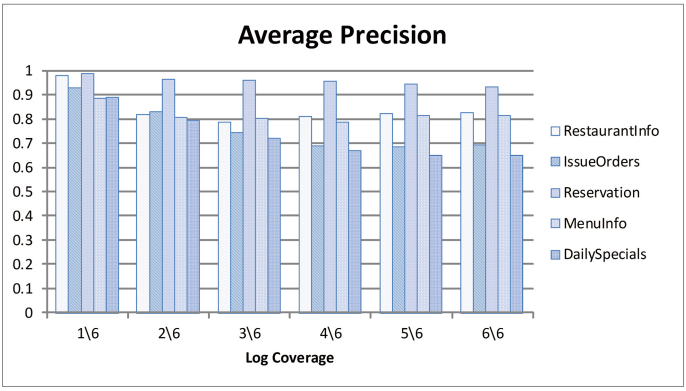
**Core.** Once the designer has determined the correct versions of the processes of the input features, we performed a set of experiments in order to compare them with the models produced by our approach with a different ratio of the coverage of the input traces. We define $P_{tool}$ as the Petri Net produced by our approach, $P_{exp}$ as the correct Petri Net and their corresponding set of valid traces as $T(P_{tool})$ and $T(P_{exp})$. Then we define $recall = \frac{|tp|}{|tp+fn|} = \frac{|T(P_{tool}) \cap T(P_{exp})|}{|T(P_{exp})|}$ and $precision = \frac{|tp|}{|tp+fp|} = \frac{|T(P_{tool}) \cap T(P_{exp})|}{|T(P_{tool})|}$ metrics where $tp$ (true positive) is the set of traces identified by our technique that are also included into the correct model, $fp$ (false positive) is the set of traces identified by our technique that are not included into the correct model, while $fn$ (false negative) is the set of traces not identified by our technique but included into the correct model. Given that the set of traces for a Petri Net could be infinite, we consider as an approximation the minimum number of traces that covers the Petri Net where loops are iterated at most once. Since ProM tool already provides a plug-in for evaluating precision and recall between two Heuristic Net's and to translate a Petri Net to a Heuristic Net, we adopted both for evaluating mined models.

For each feature we have mined different processes starting from a variable set of input traces. We have considered as input all the combinations of $1, ..., t$ traces from the global set of $t$ traces (in our experiments $t$=6). For each combination of traces we have mined the corresponding process model and we have measured precision and recall values of this model with respect to the correct one [26]. Finally, we have evaluated the average precision and recall obtained with all the combinations of 1 trace from the $t$ traces, all the combinations of 2 traces from $t$ traces, ..., and all the combinations of $t$ traces from $t$ traces. Then we repeated the same set of experiments for all features.

Figure 7(a) and (b) show the averages recall and precision values obtained for the input features. Figure 7(a) shows that for all processes the averages recall measure increases if we consider a greater set of traces as input. In case we consider all the traces, we have a recall equal to 1 meaning that all the traces




(a)




(b)

**Fig. 7.** e-Restaurant case study: average recall measure (a) and average precision measure (b) of the mined process models depending on log coverage (1-trace logs, 2-trace logs, ..., 6-trace logs).

within the correct model have been identified by our approach. On average, if the number of traces decreases, the recall decreases as well. As shown in Fig. 7(b), averages precision measure increases with the reductions of traces considered as input, meaning that on average with a lower number of traces the noise introduced by our approach is lower than with a greater number of traces. We claim that the trend of precision and recall averages do not depend on the nature of processes, indeed they all follow the same behavior which depends on the coverage of the input log.

### 3.4 Threats to Validity

Our technique extracts data-manipulation process models that may suffer of two different types of noise. The first belongs to the adopted mining algorithm

and it results in non-correct mined models having possibly additional traces. To mitigate this noise we asked designers to perform a correction to the models mined by our technique. In this way we were able to create models that were 100 % correct which have been exploited as input to evaluate the sensivitity of our technique depending on the log coverage. The second type of noise belongs to the input SQL statements that refer to technical implementation details not relevant for the application logic. To mitigate this problem, we have introduced a pre-processing phase to filter out queries that do not refer to a certain subset of the conceptual schema as selected by the designer. Our technique may also have threats to the scalability depending on the increasing input of SQL statements. Indeed, even once the non-relevant queries have been pruned out, we could mine a non-readable process due to the large space of extracted data-manipulation functions. To mitigate this problem, we advice designers to prune iteratively the conceptual schema until they obtain a readable process (by iteratively applying our technique to the input set of queries).

Recall and precision values obtained for the different features depend on the adopted mining algorithm. In [23] different mining algorithms have been compared to identify the one that better fits the application needs. In our experiments we exploited the ILP miner algorithm which is able to create models with 100 % of fitness (precision) and an acceptable level of appropriateness (recall) with respect to the input set of traces. By adopting mining algorithms with different fitness and appropriateness, we would have obtained different precision and recall values. Nevertheless, we claim that we expect to obtain similar trends of precision and recall depending on the log coverage. Indeed, we expect that by increasing the log coverage, we consequently increase the recall while lowering the precision.

The e-restaurant system adopted in the experiments can be considered a good representative for data-intensive systems i,.e., systems where most of the complexity is hidden into its interactions with a database. Indeed, the e-restaurant consists of numerous interactions with a database where data are the basis for supporting the core business activities. Even though the experimented system is of limited size and complexity, it sufficiently evidenced the capability of the proposed approach in revealing heterogeneous data-manipulation processes in real environments. With this aim, we have also mitigated the quality of the input traces by analyzing features of different nature.

## 4 Related Work

In the literature different approaches use dynamic analysis of SQL queries with a different goal than data manipulation behavior understanding. The approaches presented in [7,8] analyze SQL statements in support to database reverse engineering, e.g., detecting implicit schema constructs [8] and implicit foreign keys [7]. The approach presented by Di Penta et al. [9] identifies services from SQL traces. The authors apply FCA techniques to name services I/O parameters thus supporting the migration towards Service Oriented Architecture. Debusmann et al. [10]

present a dynamic analysis method for system performance monitoring, i.e., measuring the response time of queries sent to a remote database server. Yang et al. [11] support the recovery of a feature model by means of analyzing SQL traces. Although the former approaches analyze (particular aspects of) the data access behavior of running programs, none of the former approaches [7–11] is able to produce process models expressing such a behavior at a high abstraction level, as we do in this paper.

Other approaches (e.g., [27,28]) extract business processes by exploiting/combining static and dynamic analysis techniques, but they are not designed to deal with dynamically generated SQL queries. The most related approach, by Alalfi et al. [29], extracts scenario diagrams and UML security models by considering runtime database interactions and the state of the PHP program. These models are used for verifying security properties but they do not describe the generic data manipulation behavior of the program, they only analyze web-interface interactions. In addition they have not considered different possible instances of a given scenario as we claim it is necessary to extract a complete and meaningful model. Understanding processes starting from a set of execution traces is at the core of process mining. This paper does not make any additional contributions as far as process mining is concerned, but it is the first to apply such techniques to analyze program-database interactions.

## 5 Conclusions and Future Work

Our paper presented a tool-supported approach to recover the data manipulation behavior of data-intensive systems. The approach makes use of clustering, conceptualization and process mining techniques starting from SQL execution traces captured at runtime. We discussed how we exploited current practice technologies to implement our approach and we carried out a set of experiments to assess the quality of the mined processes depending on the coverage of the input traces. We assumed that designers were able to produce correct models based on which we measured precision and recall metrics with respect to models produced with our approach. Results showed that average precision and recall depend on the log coverage almost independently from the extracted process. As for future work we plan to enrich the input traces with multiple sources of information like user input, source code and query results with the aim of identifying the conditions that characterize decision points within process models.

## References

1. Silva, J.C., Campos, J.C., Saraiva, J.: Gui inspection from source code analysis. ECEASST **33** (2010)
2. Petit, J.M., Kouloumdjian, J., Boulicaut, J.F., Toumani, F.: Using queries to improve database reverse engineering. In: Loucopoulos, P. (ed.) ER 1994. LNCS, vol. 881, pp. 369–386. Springer, Heidelberg (1994)
3. Willmor, D., Embury, S.M., Shao, J.: Program slicing in the presence of a database state. In: ICSM 2004, pp. 448–452 (2004)

4. Cleve, A., Henrard, J., Hainaut, J.L.: Data reverse engineering using system dependency graphs. In: WCRE 2006, 157–166 (2006)
5. van den Brink, H., van der Leek, R., Visser, J.: Quality assessment for embedded sql. In: SCAM, pp. 163–170 (2007)
6. Cornelissen, B., Zaidman, A., van Deursen, A., Moonen, L., Koschke, R.: A systematic survey of program comprehension through dynamic analysis. IEEE Trans. Softw. Eng. **35**(5), 684–702 (2009)
7. Cleve, A., Meurisse, J.R., Hainaut, J.L.: Database semantics recovery through analysis of dynamic sql statements. J. Data Semant. **15**, 130–157 (2011)
8. Cleve, A., Noughi, N., Hainaut, J.-L.: Dynamic program analysis for database reverse engineering. In: Lämmel, R., Saraiva, J., Visser, J. (eds.) GTTSE 2011. LNCS, vol. 7680, pp. 297–321. Springer, Heidelberg (2013)
9. Grosso, C.D., Penta, M.D., de Guzmán, I.G.R.: An approach for mining services in database oriented applications. In: CSMR. pp. 287–296 (2007)
10. Debusmann, M., Geihs, K.: Efficient and transparent instrumentation of application components using an aspect-oriented approach. In: Brunner, M., Keller, A. (eds.) DSOM 2003. LNCS, vol. 2867, pp. 209–220. Springer, Heidelberg (2003)
11. Yang, Y., Peng, X., Zhao, W.: Domain feature model recovery from multiple applications using data access semantics and formal concept analysis. In: WCRE, pp. 215–224 (2009)
12. van der Aalst, W.M.P.: Process mining: overview and opportunities. ACM Trans. Manage. Inf. Syst. **3**(2), 7 (2012)
13. Hainaut, J.L., Henrard, J., Englebert, V., Roland, D., Hick, J.M.: Database reverse engineering. In: Liu, L., Ozhu, M.T. (eds.) Encyclopedia of Database Systems, pp. 723–728. Springer, US (2009)
14. Cleve, A., Hainaut, J.L.: What do foreign keys actually mean? In: 2012 19th Working Conference on Reverse Engineering (WCRE), pp. 299–307 (2012)
15. Kazato, H., Hayashi, S., Kobayashi, T., Oshima, T., Okada, S., Miyata, S., Hoshino, T., Saeki, M.: Incremental feature location and identification in source code. In: CSMR, pp. 371–374 (2013)
16. Alalfi, M., Cordy, J., Dean, T.: WAFA: fine-grained dynamic analysis of web applications. In: WSE 2009, pp. 41–50 (2009)
17. Ngo, M.N., Tan, H.B.K.: Applying static analysis for automated extraction of database interactions in web applications. Inf. Softw. Technol. **50**(3), 160–175 (2008)
18. Cleve, A., Hainaut, J.L.: Dynamic analysis of SQL statements for data-intensive applications reverse engineering. In: WCRE 2008, 192–196 (2008)
19. Ammons, G., Bodík, R., Larus, J.R.: Mining specifications. In: ACM Sigplan Notices, vol. 37, pp. 4–16 (2002)
20. Ganter, B., Wille, R., Wille, R.: Formal Concept Analysis. Springer, Heidelberg (1999)
21. Cleve, A., Brogneaux, A.F., Hainaut, J.L.: A conceptual approach to database applications evolution. In: ER, pp. 132–145 (2010)
22. Rozinat, A., van der Aalst, W.M.P.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. **33**(1), 64–95 (2008)
23. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: On the role of fitness, precision, generalization and simplicity in process discovery. In: Meersman, R., Panetto, H., Dillon, T., Rinderle-Ma, S., Dadam, P., Zhou, X., Pearson, S., Ferscha, A., Bergamaschi, S., Cruz, I.F. (eds.) OTM 2012, Part I. LNCS, vol. 7565, pp. 305–322. Springer, Heidelberg (2012)

24. van derWerf, J.M.E., van Dongen, B.F., Hurkens, C.A., Serebrenik, A.: Process discovery using integer linear programming. Fundamenta Informaticae **94**(3), 387–412 (2009)
25. Peterson, J.L.: Petri Net Theory and the Modeling of Systems. Prentice Hall PTR, Upper Saddle River, NJ (1981)
26. Mori, M., Noughi, N., Cleve, A.: Experiment artifacts: database schemas, sql-statements traces, data-manipulation traces and processes. http://info.fundp.ac.be/~mmo/MiningSQLTraces/
27. Nezhad, H.R.M., Saint-Paul, R., Casati, F., Benatallah, B.: Event correlation for process discovery from web service interaction logs. VLDB **20**(3), 417–444 (2011)
28. Labiche, Y., Kolbah, B., Mehrfard, H.: Combining static and dynamic analyses to reverse-engineer scenario diagrams. In: ICSM, pp. 130–139 (2013)
29. Alalfi, M.H., Cordy, J.R., Dean, T.R.: Recovering Role-Based Access Control Security Models from Dynamic Web Applications. In: Brambilla, M., Tokuda, T., Tolksdorf, R. (eds.) ICWE 2012. LNCS, vol. 7387, pp. 121–136. Springer, Heidelberg (2012)