Query Rewriting for Rule Mining in Databases

Brice Chardin[1,2], Emmanuel Coquery[1,3], Benjamin Gouriou[4], Marie Pailloux[4], and Jean-Marc Petit[1,2]

[1] Université de Lyon, CNRS
LIRIS, UMR 5205, France
[2] INSA Lyon, France
[3] Université Claude Bernard Lyon 1
[4] Université Blaise Pascal, CNRS
LIMOS, UMR 6158, Clermont-Ferrand, France

Abstract. Promoting declarative approaches in data mining is a long standing theme. This paper goes into this direction by proposing a well-founded logical query language, $Safe\mathcal{RL}$, allowing the expression of a wide variety of "rules" to be discovered against the data. $Safe\mathcal{RL}$ extends and generalizes functional dependencies in databases to new and unexpected rules easily expressed with a SQL-like language. In this setting, every rule mining problem turns out to be seen as a query processing problem. We provide a query rewriting technique and a constructive proof of the main query equivalence theorem, leading to an efficient query processing technique. Based on a concrete SQL-like grammar for $Safe\mathcal{RL}$, we show how a tight integration can be performed on top of any DBMS. The approach has been implemented and experimented on sensor network data. This contribution is an attempt to bridge the gap between pattern mining and databases and facilitates the use of data mining techniques by SQL-aware analysts.

Keywords: Pattern mining, databases, query languages, query processing

1 Introduction

The relational database management systems (DBMS) market is already huge and continues to grow since it is expected to nearly double by 2016 [1]. As a trivial consequence for the data mining community, it makes sense – more than ever – to query the data where they are by using state of the art database technologies.
While a lot of techniques have been proposed over the last 20 years for pattern mining, only a few of them are tightly coupled with a DBMS. Most of the time, some pre-processing has to be performed before the use of pattern mining techniques and the data have to be formatted and exchanged between different systems, turning round-trip engineering into a nightmare.

In this paper, we provide a logical view for a certain class of pattern mining problems. More precisely, we propose a well-founded logical query language,

EMP	Empno	Lastname	Work dept	Job	Educ level	Sex	Sal	Bonus	Comm	Mgrno
	10	SPEN	C01	FINANCE	18	F	52750	500	4220	20
	20	THOMP	B01	MANAGER	18	M	41250	800	3300	
	30	KWAN	C01	FINANCE	20	F	38250	500	3060	10
	50	GEYER	B01	MANAGER	16	M	40175	800	3214	20
	60	STERN	D21	SALE	14	M	32250	500	2580	30
	70	PULASKI	D21	SALE	16	F	36170	700	2893	100
	90	HENDER	D21	SALE	17	F	29750	500	2380	10

DEPT	Deptno	Deptname	Mgrno	Admrdept	Loc
	A00	SPIFFY CS DIV.	-	A00	-
	B01	PLANNING	20	A00	501
	C01	INF. CENTER	30	A00	403
	D01	DEV. CENTER	60	A00	-
	D11	MANUFACTURING SYSTEMS	-	D01	-
	D21	ADMIN. SYSTEMS	70	D01	501

Fig. 1. Running example

$SafeR\mathcal{L}$, based on tuple relational calculus (TRC), allowing the expression of a wide variety of "rules" to be discovered against the data. $SafeR\mathcal{L}$ extends and generalizes functional dependencies (FDs) in databases to new and unexpected rules easily expressed with a practical SQL-like language derived from $SafeR\mathcal{L}$, called $\mathcal{RQL}$. To start with, let us consider the running example given in Figure 1 with two relations *Emp* and *Dept*. *Educlevel* represents the number of years of formal education, *Sal* the yearly salary, *Bonus* the yearly bonus and *Comm* the yearly commission. This example will be used throughout the paper.

Intuitively, a $\mathcal{RQL}$ query is defined by the FINDRULES clause and generates rules of the form $X \rightarrow Y$ with X and Y disjoint attribute sets taken from the OVER clause. The SCOPE clause defines tuple-variables over some relations obtained by classical SQL queries and the CONDITION clause defines the predicate to be evaluated on each attribute occurring in $X \cup Y$.

Example 1. To make things concrete, we give some examples of $\mathcal{RQL}$ queries.

Q_1:
```
FINDRULES
OVER Empno,Lastname,Workdept,Job,Sex,Bonus
SCOPE t1,t2 Emp
CONDITION ON A IS t1.A = t2.A;
```
Q_1':
```
FINDRULES
OVER Empno,Lastname,Workdept,Job,Sex,Bonus
SCOPE t1,t2 (SELECT * FROM Emp WHERE Educlevel>16)
CONDITION ON A IS t1.A = t2.A;
```
Q_1'':
```
FINDRULES
OVER Educlevel,Sal,Bonus,Comm
SCOPE t1,t2 Emp
CONDITION ON A IS 2*ABS(t1.A-t2.A)/(t1.A+t2.A)<0.1;
```

Q_1 discovers FDs from *Emp* over a subset of attributes. Recall that a FD $X \to Y$ holds in a relation r if for all tuples $t1, t2 \in r$, and for all $A \in X$ such that $t1[A] = t2[A]$ then for all $A \in Y$, $t1[A] = t2[A]$. As shown in Q_1, $\mathcal{RQL}$ can express the discovery of FDs with a natural syntax. For example, $Empno \to Lastname$, $Workdept \to Job$ hold in *Emp*.

We can easily restrict FDs to some subset of tuples as shown with Q_1' which discovers rules comparable to conditional functional dependencies [2] by considering only employees with a level of qualification above 16. For instance, $Sex \to Bonus$ holds, meaning that above a certain level of qualification (16), the gender determines the bonus. This rule was not elicited by Q_1 because of employees 60 and 70.

Q_1'' is an approximation of FD for numeric values, where strict equality is discarded to take into account variations under 10%. For instance, salaries 41250 and 38250 are considered close (7.5% difference), but not salaries 41250 and 36170 (13.1% difference). $Sal \to Comm$ then holds, meaning that employees earning similar salaries receive similar commissions.

Nevertheless, $\mathcal{RQL}$ can do much more and is not restricted to FD at all.

Example 2. null values in *Dept*.

Q_2:
```
FINDRULES
OVER Deptname,Mgrno,Admrdept,Loc
SCOPE t1 Dept
CONDITION ON A IS t1.A IS NULL;
```

This query discovers rules between null values of the relation *Dept*. In dirty databases, null values can be a nightmare and knowing relationships between attributes with respect to null values could definitely be useful. For instance, $Mgrno \to Loc$ holds in *Dept*.

In this setting, every rule mining problem can be easily specified and turns out to be seen as a query processing problem. We provide a query rewriting technique and a constructive proof of the main query equivalence theorem, leading to an efficient query processing technique. Based on a concrete SQL-like grammar for $Safe\mathcal{RL}$, we have shown how a tight integration can be performed on top of any DBMS. The approach has been implemented and experimented on sensor network data, showing that our approach allows the discovery of meaningful rules and scales well. This contribution is an attempt to bridge the gap between pattern mining and databases to facilitate the use of data mining techniques by SQL-aware analysts. The ultimate goal of this work is to integrate pattern mining techniques into core DBMS technologies.

Related works Defining specific languages for pattern mining is a long standing goal and poses several challenges to first specify the data of interest and second, to pose pattern mining queries against these data. A survey of data mining query languages has been done in [3]. Recently, constraint programming appears to be convenient and useful since it allows addressing both issues with a unique declarative language [4, 5].

Nevertheless, we argue that pattern mining languages should benefit from direct extensions of SQL languages as done in [6], since data are most of the time stored in DBMSs. Practical approaches, as close as possible of DBMSs, have been proposed for example in [7–9] to interact more directly with DBMSs query engines. The $SafeRL$ language proposed in this paper goes into this direction by providing a formal semantic based on the TRC language, its practical counterpart $\mathcal{RQL}$ turning out to be easily implemented on top of every DBMS. FDs, association rules with 100% confidence, ad-hoc language proposed in [10] are special cases of our $Safe\mathcal{RL}$ language but none of them has a logical query language foundation.

With respect to [11], we consider a database instead of a single relation in $Safe\mathcal{RL}$, we provide a SQL-like syntax $\mathcal{RQL}$ and we focus on query processing techniques that can be reused to efficiently execute $\mathcal{RQL}$ queries. Theoretical results on decidability problems on variant of $Safe\mathcal{RL}$ languages are given in [11]. From a technical point of view, this paper is a generalization of the approach taking into account FD only [12] to compute agree sets from database relations using SQL. FDs, CFDs in databases and implications in formal concept analysis have been studied in e.g. [12–15].

Paper organization Section 2 introduces some notations and recalls important notions on relational calculus and closure systems. Section 3 presents the syntax and semantics of the $Safe\mathcal{RL}$ language, while section 4 presents some results used for computing the answer to $Safe\mathcal{RL}$ queries. Section 5 presents experimental results and section 6 concludes. Due to space constraints, proofs are omitted.

2 Preliminaries

This section introduces main definitions and notations used throughout the paper for the relational model, safe TRC, rules and closure systems.

2.1 Relational model

We use the named perspective of the relational model in which tuples are functions.

Fix a finite universe $\mathbb{U}$ of *attributes* (denoted by $\mathtt{A}, \mathtt{B}, \dots$), a countably infinite domain $\mathbb{D}$ of *constants* (denoted by $\mathtt{c}, \mathtt{c}', \dots$) and a finite set $\mathbb{R}$ of *relation symbols* (denoted by $\mathtt{R}, \mathtt{S}, \dots$). $\mathbb{U}, \mathbb{D}, \mathbb{R}$ are pairwise disjoints. Each relation symbol $\mathtt{R}$ has a schema, a subset of $\mathbb{U}$, denoted by the symbol itself, i.e. $\mathtt{R} \subseteq \mathbb{U}$. Conveniently, we will sometimes omit to refer to the relation symbol when dealing with a subset of attributes, i.e. a schema. A *tuple* $\mathtt{t}$ over $\mathtt{R}$ is a total function $\mathtt{t} : \mathtt{R} \rightarrow \mathbb{D}$. A *relation* $\mathtt{r}$ over $\mathtt{R}$ is a finite set of tuples over $\mathtt{R}$. A *database schema* $\mathbf{R}$ is a set of relation symbols, e.g. $\mathbf{R} = \{\mathtt{R_1}, \dots, \mathtt{R_n}\}$. A *database instance* (or simply a database) is a function d from $\mathbf{R}$ to the set of possible relations such that $d(\mathtt{R_i}) = \mathtt{r_i}, \mathtt{r_i}$ a relation over $\mathtt{R_i}$, for $i = 1..n$.

2.2 Variables and assignments

$SafeRL$ has different formal variables for attributes, tuples and schemata: a set $\mathbb{A}$ of attribute-variables $(A, B, \ldots)$, a set $\mathbb{T}$ of tuple-variables $(s, t, \ldots)$ and a set $\mathbb{S}$ of schema-variables $(X, Y, \ldots)$. $\mathbb{A}, \mathbb{T}, \mathbb{S}, \mathbb{U}, \mathbb{D}, \mathbb{R}$ are pairwise disjoints.

An attribute-assignment ρ (resp. a schema-assignment Σ) is a function that maps an attribute-variable A (resp. a schema-variable X) to an attribute $\rho(A) \in \mathbb{U}$ (resp. a subset of attributes $\Sigma(X) \subseteq \mathbb{U}$). A tuple-assignment σ is also a function from a tuple-variable t to a tuple $\mathtt{t}$ defined over some schema. Conveniently, a tuple-variable t can be explicitly defined over some schema $\mathtt{X}$, noted by $t : \mathtt{X}$ and we will use the notation $sch(t) = \mathtt{X}$.

For an attribute-assignment ρ (as well as for tuple-assignments and schema-assignments) we denote by $\rho_{A \mapsto \mathtt{A}}$ the assignment defined by:

$$\rho_{A \mapsto \mathtt{A}}(B) = \begin{cases} \mathtt{A} & \text{if } B = A \\ \rho(B) & \text{if } B \neq A \end{cases}$$

2.3 Safe TRC

For the sake of clarity, we recall here the syntax and semantics of the TRC in its simplest form. TRC formulas noted $\psi, \psi_1, \psi_2, \ldots$ are defined inductively as usual, where $\mathtt{A}, \mathtt{B} \in \mathbb{U}$, $\mathtt{X} \subseteq \mathbb{U}$, $\mathtt{c} \in \mathbb{D}$, $\mathtt{R} \in \mathbb{R}$, $t, t_1, t_2 \in \mathbb{T}$:

$$\mathtt{R}(t) \mid t_1.\mathtt{A} = t_2.\mathtt{B} \mid t.\mathtt{A} = \mathtt{c} \mid \neg\psi \mid \psi_1 \wedge \psi_2 \mid \exists t : \mathtt{X}\ (\psi)$$

Given a database d over $\mathbf{R}$ and a tuple assignment σ, the satisfaction of a TRC formula ψ is inductively defined as follows:

- $\langle d, \sigma \rangle \models \mathtt{R}(t)$ if $\sigma(t) \in d(\mathtt{R}), \mathtt{R} \in \mathbf{R}$
- $\langle d, \sigma \rangle \models t_1.\mathtt{A} = t_2.\mathtt{B}$ if $\sigma(t_1)(\mathtt{A}) = \sigma(t_2)(\mathtt{B})$
- $\langle d, \sigma \rangle \models t.\mathtt{A} = \mathtt{c}$ if $\sigma(t)(\mathtt{A}) = \mathtt{c}$
- $\langle d, \sigma \rangle \models \neg\psi$ if $\langle d, \sigma \rangle \not\models \psi$
- $\langle d, \sigma \rangle \models \psi_1 \wedge \psi_2$ if $\langle d, \sigma \rangle \models \psi_1$ and $\langle d, \sigma \rangle \models \psi_2$
- $\langle d, \sigma \rangle \models \exists t : \mathtt{X}\ (\psi)$ if there exists a tuple $\mathtt{t}$ over $\mathtt{X}$ such that $\langle d, \sigma_{t \mapsto \mathtt{t}} \rangle \models \psi$

A TRC query is an expression of the form

$$q = \{t : \mathtt{X} \mid \psi\}$$

where ψ is a TRC formula with exactly one free variable t. The set of answers $ans(q, d)$ of q w.r.t. a database d is

$$ans(q, d) = \{\sigma(t) \mid \langle d, \sigma \rangle \models \psi\}$$

In the sequel, we will consider a restriction of the TRC, equivalent to the relational algebra in order to be compatible with SQL, known as *safe TRC* [16].

2.4 Rules and closure systems

Rules or implications, closure systems and closure operators have been widely studied in many branches of applied mathematics and computer sciences, with many applications in databases with functional dependencies [17] and in formal concept analysis with implications [18]. The interested reader should refer to [19] for a comprehensive survey. We summarize the main results that are useful for the rest of the paper.

Let $U \subseteq \mathbb{U}$. A closure system C on U is such that $U \in C$ and for all $X, Y \in C$, $X \cap Y \in C$ [18]. Let F be a set of rules on U. A closure system can be defined for F, noted $CL(F) = \{X \subseteq U | X = X_F^+\}$ where X_F^+ is the closure of X with respect to F. Let $IRR(F)$ be the set of meet-irreducible elements of $CL(F)$. The notion of *base* of a closure system is defined as follows:

Definition 1. *Let $CL(F)$ be a closure system. A base $\mathcal{B}$ of $CL(F)$ is such that $IRR(F) \subseteq \mathcal{B} \subseteq CL(F)$*

A base is called a context in FCA terminology [18]. From a functional dependency inference point of view, we quote the following approach [20, 12] to discover the so-called canonical basis from a relation r:

1. Compute a base of the closure system associated to FDs from r (known as *agree sets*),
2. From the base, compute the unique canonical cover for exact FDs and Gottlob and Libkin cover for approximate FDs [21, 20, 22].

The rest of this paper proposes a generalization of this approach. Indeed, each $Safe\mathcal{RL}$ query defines a closure system and therefore, in order to reuse previous results, the problem turns out to be on the computation of a base with respect to a query from the database [12]. Due to space constraints, we will focus on this first stage, the second stage will be omitted.

3 A Query Language for Rule Mining

In the introduction, we have illustrated $\mathcal{RQL}$ – a SQL-like friendly language – through examples. This section formally defines the syntax and semantics of $Safe\mathcal{RL}$ from which $\mathcal{RQL}$ is derived. We have introduced safe TRC for expressing SQL like queries. Before defining $Safe\mathcal{RL}$, it remains to precisely define (cf. previous examples of $\mathcal{RQL}$ queries) the `CONDITION` clause, through the notion of *mining formulas*:

```
CONDITION ON A IS delta_cond(A, t1, ..., tn);
```

3.1 Mining Formulas

Mining formulas, denoted by $\delta, \delta_1, \delta_2, \ldots$, are defined over tuple-variables $\mathbb{T}$, attribute-variables $\mathbb{A}$ and constants $\mathbb{D}$ only. Their syntax and their semantics are defined as follows.

Definition 2. *Let $t, t_1, t_2 \in \mathbb{T}$, $A, B \in \mathbb{A}$ and $\mathtt{c} \in \mathbb{D}$. A mining formula is of the form: $t_1.A = t_2.B \mid t.A = \mathtt{c} \mid \neg\delta \mid \delta_1 \wedge \delta_2$*

The satisfaction *of a mining formula δ w.r.t. a tuple-assignment σ and an attribute-assignment ρ, denoted by $\langle\sigma, \rho\rangle \models \delta$, is inductively defined as follows:*

- $\langle\sigma, \rho\rangle \models t_1.A = t_2.B$ ***iff*** $\sigma(t_1)(\rho(A)) = \sigma(t_2)(\rho(B))$
- $\langle\sigma, \rho\rangle \models t.A = \mathtt{c}$ ***iff*** $\sigma(t)(\rho(A)) = \mathtt{c}$
- $\langle\sigma, \rho\rangle \models \neg\delta$ ***iff*** $\langle\sigma, \rho\rangle \not\models \delta$
- $\langle\sigma, \rho\rangle \models \delta_1 \wedge \delta_2$ ***iff*** $\langle\sigma, \rho\rangle \models \delta_1$ *and* $\langle\sigma, \rho\rangle \models \delta_2$

Such formulas are very simple and restrictive: given one attribute and several tuples, it allows to tell whether or not a mining formula holds.

3.2 $Safe\mathcal{RL}$ queries

The $Safe\mathcal{RL}$ query language can now be defined. A $Safe\mathcal{RL}$ query over a database schema $\mathbf{R}$ is an expression of the form:

$$Q = \{X \to Y \mid \forall t_1 \ldots \forall t_n(\psi(t_1, \ldots, t_n) \wedge (\forall A \in X(\delta(A, t_1, \ldots, t_n)) \to \forall A \in Y(\delta(A, t_1, \ldots, t_n))))\}, \text{ where:}$$

- X, Y are schema-variables
- ψ is a TRC-formula over $\mathbf{R}$ with n free tuple-variables $t_1, \ldots, t_n$
- δ is a mining formula with $t_1, \ldots, t_n$ free tuple-variables and A a free attribute-variable

When clear from context, a $Safe\mathcal{RL}$ query Q may also be simply denoted by $Q = \langle\psi(t_1, \ldots, t_n), \delta(A, t_1, \ldots, t_n)\rangle$ or even $Q = \langle\psi, \delta\rangle$.

Example 3. The mining of FDs over `EMP` is expressed as the query $Q = \langle\mathtt{EMP}(t_1) \wedge \mathtt{EMP}(t_2), (t_1.A = t_2.A)\rangle$.

The attributes of ψ are equal to $\bigcup_{i=1}^{n} sch(t_i)$ whereas the schema of Q, denoted by $sch(Q)$, is defined by: $sch(Q) = \bigcap_{i=1}^{n} sch(t_i)$: only common attributes of tuple-variables are meaningful to discover rules.

To specify the result of the evaluation of a $Safe\mathcal{RL}$ query against a database, we define the notion of satisfaction.

A $Safe\mathcal{RL}$ query $\langle\psi, \delta\rangle$ is satisfied in a database d and w.r.t. a schema-assignment Σ, denoted by $\langle d, \Sigma\rangle \models \langle\psi, \delta\rangle$ if the following holds:

For all tuple-assignment σ such that $\langle d, \sigma\rangle \models \psi$: (1)
  if for all $\mathtt{A} \in \Sigma(X)$, $\langle\sigma, \rho_{A \mapsto \mathtt{A}}\rangle \models \delta$ (2)
  then for all $\mathtt{A} \in \Sigma(Y)$, $\langle\sigma, \rho_{A \mapsto \mathtt{A}}\rangle \models \delta$ (3)

Intuitively, this definition generalizes the definition of FD satisfaction in a relation: instead of only 2 tuples, we may have n tuples from many relations and verifying a certain condition (1); and instead of the condition "for all $A \in R, t_1[A] = t_2[A]$", we may have any mining formula (2), (3).

The answer of a $Safe\mathcal{RL}$ query $Q = \langle\psi, \delta\rangle$ in a database d over $\mathbf{R}$, denoted by $ans(Q, d)$, is defined as: $ans(Q, d) = \{\Sigma(X) \to \Sigma(Y) \mid \langle d, \Sigma\rangle \models \langle\psi, \delta\rangle, \Sigma(X) \cup \Sigma(Y) \subseteq sch(Q)\}$.

3.3 $\mathcal{RQL}$: A practical language for $\boldsymbol{Safe\mathcal{RL}}$

$\mathcal{RQL}$ is a practical SQL-like declarative language to express $Safe\mathcal{RL}$ queries. Let us consider a $Safe\mathcal{RL}$ query $Q = \langle \psi(t_1, \ldots, t_n), \delta(A, t_1, \ldots, t_n)\rangle$ and its associated $\mathcal{RQL}$ query:

```
FINDRULES
OVER A1, ..., An
SCOPE t1 (SQL1), ..., tn (SQLn)
WHERE condition(t1, , ..., tn)
CONDITION ON A IS delta_cond(A, t1, ..., tn);
```

The clause FINDRULES identifies $\mathcal{RQL}$ queries. The clause SCOPE specifies every tuple to be used to discover the rules and corresponds to the tuple-variables of ψ, a safe TRC formula. The clause CONDITION gives the condition to be observed against the data, the so-called mining formula δ. The clause OVER allows restrictions of rules to a particular set of attributes, typically the attributes of $sch(Q)$. The WHERE clause is defined as usual. We have already given many examples of $\mathcal{RQL}$, other details are omitted. Note that $\mathcal{RQL}$ allows much more flexibility than $Safe\mathcal{RL}$ since syntactic sugars available in SQL can be used for free, as in query Q_1'' of Example 1.

4 Theoretical results

In [11], a slightly different language for rule mining has been proposed. One of the main results was to point out that every query is "Armstrong-compliant", meaning basically that Armstrong axioms are sound and that each query defines a closure system. The same result holds for $Safe\mathcal{RL}$ queries.

Given a database d and a $Safe\mathcal{RL}$ query Q, the basic idea is to compute a base of the closure system associated with Q from d. Let us start by introducing the closure system associated with Q.

An appendix provides the proofs of all propositions, lemmas and theorems.

4.1 Closure system and bases for $\boldsymbol{Safe\mathcal{RL}}$ queries

Given a query Q against a database d, the definitions of a base and a closure system given in Section 2.4 are extended to $ans(Q, d)$.

We say that $\mathtt{Z} \subseteq \mathbb{U}$ satisfies $ans(Q, d)$ if for all $\mathtt{X} \rightarrow \mathtt{Y} \in ans(Q, d)$, $\mathtt{X} \not\subseteq \mathtt{Z}$ or $\mathtt{Y} \subseteq \mathtt{Z}$. The *closure system* of Q in d, denoted by $CL_Q(d)$, is defined by: $CL_Q(d) = \{\mathtt{Z} \subseteq sch(Q) \mid \mathtt{Z}$ satisfies $ans(Q, d)\}$.

In our setting, the definition of the base is:

Definition 3. *Let $Q = \langle \psi, \delta\rangle$ be a $Safe\mathcal{RL}$ query over $\mathbf{R}$ and d a database over $\mathbf{R}$. The* base *of Q in d, denoted by $B_Q(d)$, is defined by:*

$$B_Q(d) = \bigcup_{\substack{\sigma \ s.t. \\ \langle d, \sigma\rangle \models \psi}} \Big\{\{\mathtt{A} \in sch(Q) \mid \langle \sigma, \rho_{A \mapsto \mathtt{A}}\rangle \models \delta\}\Big\}$$

That is, $B_Q(d)$ is the set of all $\mathtt{Z} \subseteq sch(Q)$ for which there exists σ such that $\langle d, \sigma\rangle \models \psi$ and $\langle \sigma, \rho_{A\mapsto\mathtt{A}}\rangle \models \delta$ for all $\mathtt{A} \in \mathtt{Z}$. Note that since A is the only free attribute variable in δ, using $\rho_{A\mapsto\mathtt{A}}$ fully determines the attribute assignment in the evaluation of δ.

Proposition 1. *$B_Q(d)$ is a base of the closure system $CL_Q(d)$.*

4.2 Computing the base of a *SafeRL* query using query rewriting

The naive approach consists in executing n SQL queries against the database, to cache all intermediary results, to keep only the right combination of tuples with respect to the WHERE clause and then to compute the base of the closure system. We can do much better: the basic idea is to decompose the query in order to push as much as possible the processing into the SQL query engine.

For every $\mathcal{RQL}$ query $Q = \langle\psi, \delta\rangle$ involving n tuple-variables, there exists another query $Q' = \langle\psi', \delta'\rangle$ with a *unique tuple-variable* . The practical consequence of this remark is that the computation of the base can be done in a unique SQL query, i.e. the base of $\langle\psi', \delta'\rangle$ can be delegated to the SQL query engine which was not the case in the former formulation. By means of a rewriting function rw, we transform $Q = \langle\psi, \delta\rangle$ into $Q' = \langle\psi', \delta'\rangle$. The idea of rw is that the unique tuple-variable t appearing in Q' takes its values in the schema built from the disjoint union of $sch(t_i), i = 1..n$ and is essentially the concatenation of the initial t_i's.

Let $\mathtt{R}$ be the new schema built from the disjoint union of the t_i's, i.e. $\mathtt{R} = \bigcup_{i\in 1..n}\{\langle\mathtt{A}, i\rangle \mid \mathtt{A} \in sch(t_i)\}$.

The function rw is defined on mining formulae inductively, with t a fresh tuple-variable. Each fresh attribute-variable A_i replaces A in δ', noted $\mathsf{rw}(\delta)$ in the sequel, for the corresponding t_i part of t:

$$\mathsf{rw}(\delta) = \begin{cases} \neg\mathsf{rw}(\delta') & \text{if } \delta \text{ is } \neg\delta' \\ \mathsf{rw}(\delta_1) \wedge \mathsf{rw}(\delta_2) & \text{if } \delta \text{ is } \delta_1 \wedge \delta_2 \\ t.A_i = t.A_j & \text{if } \delta \text{ is } (t_i.A = t_j.A) \\ t.A_i = \mathsf{c} & \text{if } \delta \text{ is } (t_i.A = \mathsf{c}) \end{cases}$$

We also overload rw to transform tuple-assignment σ and attribute-assignment ρ into respective assignments. $\mathsf{rw}(\sigma) = \sigma'$ is defined by $\sigma'(t) = \mathtt{t}$ and $\mathtt{t}(\langle\mathtt{A}, i\rangle) = \sigma(t_i)(\mathtt{A})$. $\mathsf{rw}(\rho) = \rho'$ is defined by $\rho'(A_i) = \langle\rho(A), i\rangle$.

Given a mining formula and the previous rewriting $\mathsf{rw}()$, we have the following lemma.

Lemma 1. $\langle\sigma, \rho\rangle \models \delta$ *iff* $\langle\mathsf{rw}(\sigma), \mathsf{rw}(\rho)\rangle \models \mathsf{rw}(\delta)$

Finally, it remains to rewrite TRC formula ψ with $\mathsf{rw}(\psi)$ by forcing t to have each t_i as one of its components:

$$\mathsf{rw}(\psi) = \exists t_1 : sch(t_1), \ldots, t_n : sch(t_n)(\psi \wedge \bigwedge_{\mathtt{A}\in sch(Q)} \bigwedge_{i=1}^{n} t.\mathtt{A}_i = t_i.\mathtt{A})$$

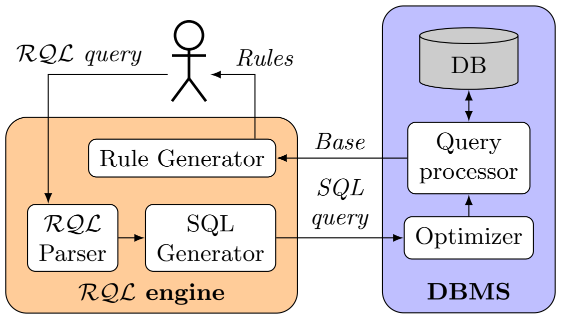


Fig. 2. Query processing overview

Given a safe TRC formula and the previous rewriting $\mathsf{rw}()$, we have the following result.

Lemma 2. $\langle d, \sigma \rangle \models \psi$ *iff* $\langle d, \mathsf{rw}(\sigma) \rangle \models \mathsf{rw}(\psi)$

Therefore $B_Q(d)$ is computable by running only one SQL query, corresponding to the safe TRC query $\{t \mid \mathsf{rw}(\psi)\}$, except for the SELECT part that consists in evaluating, for each $\mathtt{A} \in sch(Q)$, the satisfaction of $\mathsf{rw}(\delta)$.

Theorem 1.

$$B_Q(d) = \bigcup_{\substack{\mathsf{rw}(\sigma) \text{ s.t.} \\ \langle d, \mathsf{rw}(\sigma) \rangle \models \mathsf{rw}(\psi)}} \Big\{ \{ \mathtt{A} \in sch(Q) \mid \exists \rho : \rho(A) = \mathtt{A} \wedge \langle \mathsf{rw}(\sigma), \mathsf{rw}(\rho) \rangle \models \mathsf{rw}(\delta) \} \Big\}$$

The previous theorem allows pushing query processing as much as possible into the DBMS to compute the base of the closure system associated to every $Safe\mathcal{RL}$ query.

5 Implementation and experiments

Given a $\mathcal{RQL}$ query Q, the query processing engine consists of a Java/JavaCC application to:

1. Compute the base of the closure system of Q using the generated SQL query provided by Theorem 1.
2. Compute the canonical cover for exact rules and a Gottlob and Libkin cover for approximate rules [21]. Details are out of the scope of this paper, note just that we have reused the code of T. Uno [23] for the most expensive part of the rule generation process.

Figure 2 gives an overview of $\mathcal{RQL}$ query processing.

samples	
id	DECIMAL(20,0)
timestamp	TIMESTAMP
type	DECIMAL(3,0)
value	DECIMAL(10,0)

descriptions	
id	DECIMAL(20,0)
type	VARCHAR(12)
location	VARCHAR(18)
description	VARCHAR(78)

Fig. 3. PlaceLab database schema

time	bathroom_ light	kitchen_ humidity_0	...	bedroom_ temperature_5
2006-08-22 00:00:00	0.4971428	4344	...	21.43
2006-08-22 00:01:00	0.6685879	4344	...	21.43
2006-08-22 00:02:00	0.4985673	4344	...	21.465
		...		
2006-09-18 23:58:00	1567.7822	5324	...	22.53
2006-09-18 23:59:00	1563.5891	5276	...	22.50

Fig. 4. Sensors data

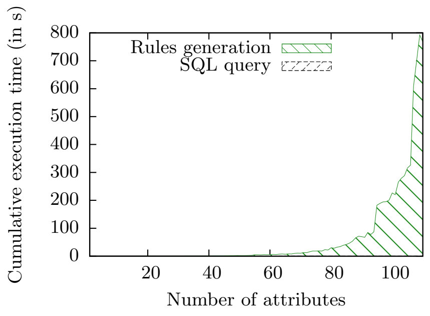
5.1 Sensor Data

We experimented our $\mathcal{RQL}$ processing engine using the PlaceLab dataset provided by the MIT House_n Consortium and Microsoft Research [24].

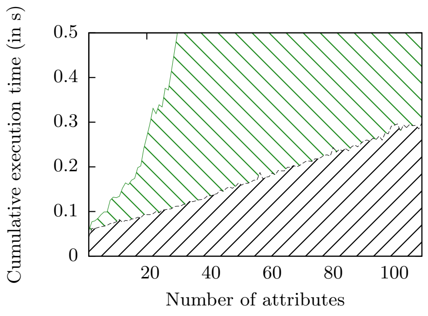
The PlaceLab is a 93 m^2 apartment instrumented with several hundred sensors. During the experiment, interior conditions (temperature, humidity, light, pressure, current, gas and water flow) were captured by 106 sensors, along with 92 reed switches installed on doors, drawers and windows to detect open-closed events. 277 wireless motion sensors were placed on nearly all objects that might be manipulated. Two participants used the PlaceLab as a temporary home for 10 weeks.

The available data is a subset of about a month from the original 2.5 months experiment, from August 22, 2006 to September 18. Raw data is extracted from binary files, where each reading contains a sensor id, a timestamp and a value (the measurement). Sensors meta-data include, for each sensor id, type, location and a short textual description, along with other meta-data of little interest for our experiments, such as installation date, etc. This data is stored in an Oracle database whose schema is depicted in figure 3.

For data mining queries, we focused on variations of the physical properties of the environment, such as temperature, light, etc., which amount to more than 100 million tuples. A view, easily expressed with SQL, has been created to synchronize and resample every sensor with a common sampling interval (one minute). This view, illustrated in figure 4, has 165 attributes and 32543 tuples. Except for the time, each attribute is associated either with one of the 106 physical properties sensors or one of the 58 selected switches.



(a) Rule generation



(b) Zoom on SQL queries durations

Fig. 5. Execution time for null rules

5.2 Experimental Results

The server used for these experiments is a virtual machine running on VMware 4.1 with 2× Intel Xeon X5650 and 7.2K disks in RAID 5. This virtual machine, running Debian 6.0, disposes of 16 GB of RAM and 4 CPU cores. The DBMS is Oracle Database 11g Release 2.

In these experiments, we consider three families of $\mathcal{RQL}$ queries.

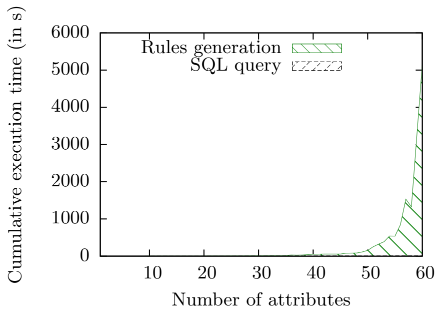
Q1: rules for null values The first set of queries mine rules between sensors for null values. Such queries can be used to identify groups of sensors which are unavailable simultaneously, due, for example, to a shared acquisition system.

```
FINDRULES
OVER <list of attributes>
SCOPE t1 sensors
CONDITION ON A IS t1.A IS NULL;
```

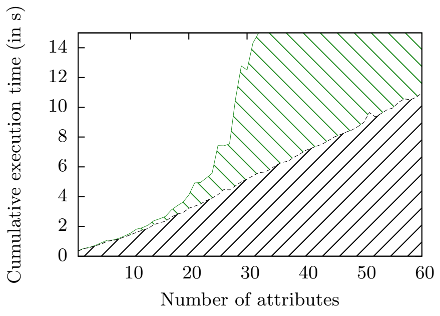
For example, if we consider all temperature sensors as the list of attributes, we can group sensors dining_room_temperature_1 (A), dining_room_temperature_2 (B), dining_room_temperature_3 (C), dining_room_temperature_4 (D), hall_temperature_0 (E) and hall_temperature_3 (F) according to rules ($F \to C$, $A \to CF$, $AD \to E$, $BE \to D$, $EF \to A$, $AB \to DE$, $CE \to AF$, $DF \to AE$, $BF \to ADE$, $CD \to AEF$, $BC \to ADEF$). Similarly, we can group four sensors from the living room, two sensors from the office, two sensors from the hall with three sensors from the kitchen, etc.

Figure 5 gives the cumulative execution time for various number of attributes in the *OVER* clause of Q1. As expected, rule generation is by far the most expensive part when the query runs over a large set of attributes. The SQL query however lasts less than a second and increases linearly: computation of the base by the DBMS is efficient.

Q2: Functional dependencies The second set of queries finds functional dependencies within a time window. This time window is specified using SQL conditions on the timestamp.



(a) Rule generation



(b) Zoom on SQL queries durations

Fig. 6. Execution time for functional dependencies

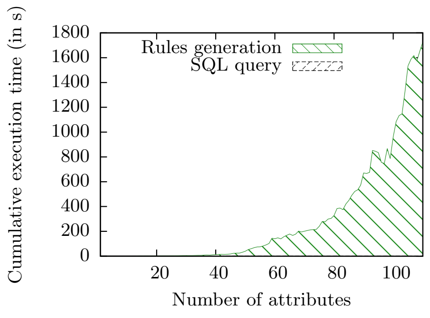
```
FINDRULES
OVER <list of attributes>
SCOPE t1,t2 (
  SELECT * FROM sensors
  WHERE time BETWEEN '2006-08-31 00:00:00'
                 AND '2006-08-31 23:59:59')
WHERE t1.rowid < t2.rowid
CONDITION ON A IS t1.A = t2.A;
```

To generate the base, the corresponding SQL query performs a Cartesian product (more precisely, a theta self-join on t1.rowid < t2.rowid, which gives half as many tuples). Consequently, the SQL part is significantly more costly compared with the previous family of $\mathcal{RQL}$ queries. Figure 6 gives the cumulative execution time for various number of attributes in the *OVER* clause of Q2.

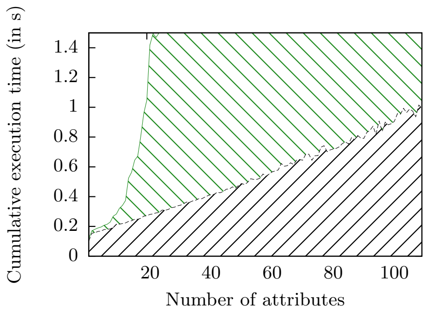
Q3: rules for local maximums $\mathcal{RQL}$ queries can express a wide range of conditions for rules. For example, the following query finds rules for local maximums of measurements (i.e. $X \to A$ is interpreted as: if all sensors in X have a local maximum at time T, then sensor A has a local maximum at time T).

```
FINDRULES
OVER <list of attributes>
SCOPE t1,t2,t3 sensors
WHERE t2.time = t1.time+interval '1' minute
  AND t3.time = t2.time+interval '1' minute
CONDITION ON A IS t1.A < t2.A AND t2.A > t3.A;
```

Even though this query has three tuple-variables, all three are bound by a 1:1 relationship. Consequently, this query is computed efficiently. Figure 7 shows performances similar to Q1.



(a) Rule generation



(b) Zoom on SQL queries durations

Fig. 7. Execution time for local maximums rules

6 Conclusion

In this paper, we have introduced $SafeRL$, a logical query language based on TRC and devoted to rule discovery in databases. The rule mining problem is seen as a query processing problem, for which we have proposed a query rewriting technique allowing the delegation of as much processing as possible to the underlying DBMS engine. $\mathcal{RQL}$, the concrete language of $Safe\mathcal{RL}$, is a SQL-like pattern mining language which allows SQL developers to extract precise information without specific knowledge in data mining. A system has been developed and tested against a real-life database provided by the MIT House_n Consortium and Microsoft Research [24]. These experiments show both the feasibility and the efficiency of the proposed language. A web prototype for $\mathcal{RQL}$ has been released and is available for teaching and research: `http://rql.insa-lyon.fr`

References

1. MarketResearch.com: Worldwide relational database management systems 2012-2016 forecast (Aug 2012)
2. Bohannon, P., Fan, W., Geerts, F., Jia, X., Kementsietsidis, A.: Conditional functional dependencies for data cleaning. In: Proceedings of the 23rd International Conference on Data Engineering. ICDE '07 (2007) 746–755
3. Blockeel, H., Calders, T., Fromont, E., Goethals, B., Prado, A., , Robardet, C.: A practical comparative study of data mining query languages. In Springer, ed.: Inductive Databases and Constraint-Based Data Mining. Sao Deroski, Bart Goethals, Pane Panov (2010) 59–77
4. Guns, T., Nijssen, S., Raedt, L.D.: Itemset mining: A constraint programming perspective. Artif. Intell. **175**(12-13) (2011) 1951–1983
5. Métivier, J.P., Boizumault, P., Crémilleux, B., Khiari, M., Loudni, S.: A constraint-based language for declarative pattern discovery. In: Declarative Pattern Mining Workshop, ICDM 2011. (2011) 1112–1119
6. Calders, T., Wijsen, J.: On monotone data mining languages. In: Proceedings of the 8th International Workshop on Database Programming Languages. DBPL '01 (2001) 119–132

7. Fang, L., LeFevre, K.: Splash: ad-hoc querying of data and statistical models. In: Proceedings of the 13th International Conference on Extending Database Technology. EDBT '10 (2010) 275–286
8. Ordonez, C., Pitchaimalai, S.K.: One-pass data mining algorithms in a DBMS with UDFs. In: SIGMOD Conference. (2011) 1217–1220
9. Blockeel, H., Calders, T., Fromont, É., Goethals, B., Prado, A., Robardet, C.: An inductive database system based on virtual mining views. Data Min. Knowl. Discov. **24**(1) (2012) 247–287
10. Agier, M., Petit, J.M., Suzuki, E.: Unifying framework for rule semantics: Application to gene expression data. Fundam. Inform. **78**(4) (2007) 543–559
11. Agier, M., Froidevaux, C., Petit, J.M., Renaud, Y., Wijsen, J.: On Armstrong-compliant logical query languages. In: Proceedings of the 4th International Workshop on Logic in Databases. LID '11 (2011) 33–40
12. Lopes, S., Petit, J.M., Lakhal, L.: Functional and approximate dependency mining: database and FCA points of view. J. Exp. Theor. Artif. Intell. **14**(2-3) (2002) 93–114
13. Medina, R., Nourine, L.: A unified hierarchy for functional dependencies, conditional functional dependencies and association rules. In: ICFCA. Volume 5548 of LNCS., Springer (2009) 98–113
14. Medina, R., Nourine, L.: Conditional functional dependencies: An FCA point of view. In: ICFCA. (2010) 161–176
15. Babin, M.A., Kuznetsov, S.O.: Computing premises of a minimal cover of functional dependencies is intractable. Discrete Applied Mathematics **161**(6) (2013) 742–749
16. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
17. Armstrong, W.W.: Dependency structures of data base relationships. In: Proceedings of the IFIP Congress. (1974) 580–583
18. Ganter, B., Wille, R.: Formal Concept Analysis. Springer (1999)
19. Caspard, N., Monjardet, B.: The lattices of closure systems, closure operators, and implicational systems on a finite set: A survey. Discrete Applied Mathematics **127**(2) (2003) 241–269
20. Mannila, H., Räihä, K.J.: Algorithms for inferring functional dependencies from relations. Data and Knowldege Engineering **12**(1) (1994) 83–99
21. Gottlob, G., Libkin, L.: Investigations on Armstrong relations, dependency inference, and excluded functional dependencies. Acta Cybernetica **9**(4) (1990) 385–402
22. Demetrovics, J., Thi, V.D.: Some remarks on generating Armstrong and inferring functional dependencies relation. Acta Cybernetica **12**(2) (1995) 167–180
23. Murakami, K., Uno, T.: Efficient algorithms for dualizing large-scale hypergraphs. CoRR **1102.3813** (2011)
24. Intille, S.S., Larson, K., Tapia, E.M., Beaudin, J.S., Kaushik, P., Nawyn, J., Rockinson, R.: Using a live-in laboratory for ubiquitous computing research. In: PERVASIVE '06. (2006) 349–365