RQL: A Query Language for Rule Discovery in Databases

Brice Chardin, Emmanuel Coquery, Marie Pailloux, Jean-Marc Petit

▶ **To cite this version:**

Brice Chardin, Emmanuel Coquery, Marie Pailloux, Jean-Marc Petit. RQL: A Query Language for Rule Discovery in Databases. Theoretical Computer Science, Elsevier, 2017, 658.

HAL Id: hal-01395083

https://hal.archives-ouvertes.fr/hal-01395083

Submitted on 10 Nov 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

RQL: A Query Language for Rule Discovery in Databases

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

Abstract

Promoting declarative approaches in data mining is a long standing theme, the main idea being to simplify as much as possible the way data analysts interact with their data. This paper goes into this direction by proposing a well-founded logical query language, $SafeRL$, allowing the expression of a wide variety of rules to be discovered against a database. By rules, we mean statements of the form "if ... then ...", as defined in logics for "implications" between boolean variables. As a consequence, $SafeRL$ extends and generalizes functional dependencies to new and unexpected rules. We provide a query rewriting technique and a constructive proof of the main query equivalence theorem, leading to an efficient query processing technique. From $SafeRL$, we have devised RQL, a user-friendly SQL-like query language. We have shown how a tight integration can be performed on top of any relational database management system. Every RQL query turns out to be seen as a query processing problem, instead of a particular rule mining problem. This approach has been implemented and experimented on sensor network data. A web prototype has been released and is freely available (`http://rql.insa-lyon.fr`). Data analysts can upload a sample of their data, write their own RQL queries and get answers to know whether or not a rule holds (if not, a counter example from the database is displayed) and much more.

keywords Query Languages, Formal Concept Analysis, Implications, Functional dependencies, Query Optimization, Relational Calculus

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 [41]. As a trivial consequence for the data mining community, it makes sense – more

[1]LIAS, ISAE-ENSMA, France
[2]Department of Computer Science, University Lyon 1, CNRS, France
[3]Department of Computer Science, University Clermont-Ferrand 2, CNRS, France
[4]Université de Lyon, CNRS, INSA-Lyon, LIRIS, France

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	-	MANAGER	18	M	41250	800	3300	-
	30	KWAN	-	FINANCE	20	F	38250	500	3060	10
	50	GEYER	-	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
	100	SPEN	C01	FINANCE	18	M	26150	800	2092	20

Figure 1: Running example

than ever – to query the data in-place when 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, $Safe\mathcal{RL}$, based on tuple relational calculus (TRC), allowing the expression of a wide variety of rules to be discovered against the data. By rules, we mean statements of the form "if ... then ...", as defined in logics for "implications" between boolean variables.

From a database perspective, $Safe\mathcal{RL}$ extends and generalizes functional dependencies (FDs) to new and unexpected rules easily expressed with a practical SQL-like language, called RQL, derived from $Safe\mathcal{RL}$. To start with, let us consider the running example given in Figure 1 with a relation *Emp* to depict employees. *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 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, denoted by the named variable $\$A$, occurring in $X \cup Y$.

Example 1. *To make things concrete, we give some examples of 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 the subset of attributes specified in the `OVER` clause. Recall that a FD $X \rightarrow Y$ 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, RQL allows to express the discovery of FDs with a natural syntax. For example, $Empno \rightarrow Lastname$ and $Workdept \rightarrow Job$ 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 [14] by considering only employees with a level of qualification above 16. For instance, $Sex \rightarrow Bonus$ holds, meaning that above a certain level of qualification (16), the gender determines the bonus. This rule was not elicited by Q_1 because there exist several counter-examples, such as employees 30 and 70.

Q''_1 is an approximation of FDs for numeric values, similar to Metric Functional Dependencies [35], 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). $Sal \rightarrow Comm$ then holds, meaning that employees earning similar salaries receive similar commissions.

With respect to the expressiveness of the language, RQL allows to "catch" implications in their purest logic form. On the one hand, it turns out that two famous examples, namely association rules with 100% confidence and functional dependencies, can be expressed with RQL, since both are logical implications. Nevertheless, RQL can do much more, as shown in previous examples and throughout the paper (see section 1.1.). On the other hand, RQL cannot be used to express every possible rule mining problem. For instance, it has not been conceived to be equivalent to association rules, which are not strictly logical implications. Frequency requirements (e.g. support and confidence) immediately remove key properties such as reflexivity ($Y \subset X, X \rightarrow Y$ does not hold if X is not frequent).

RQL has been devised as a user-friendly SQL-like query language, which can be integrated on top of any DBMS supporting SQL. RQL query processing is seen as a classical query processing problem in databases. We provide a query rewriting technique and a constructive proof of the main query equivalence theorem, leading to an efficient query processing technique.

This approach has been implemented and experimentally evaluated on sensor network data. A web prototype has been released and is freely available (`http://rql.insa-lyon.fr`). Data analysts can upload a sample of their data, write their own RQL queries and get answers to know whether or not a rule holds (if not, a counter example from the database is displayed) and much more.

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.

1.1. More RQL examples

Conveniently, we have reused so far RQL examples related to FDs. Nevertheless, RQL does much more and is not restricted to FDs at all.

Example 2. *null values in Dept.*

Q_2:
```
FINDRULES
   OVER Empno, Lastname, Workdept, Job, Sex, Bonus,
        Mgrno
   SCOPE t1 Emp
   CONDITION ON $A IS t1.$A IS NULL
```

Q_2 discovers rules between null values of the relation *Emp*. In most databases, null values are common and knowing relationships between attributes with respect to null values could be useful. For instance, $Mgrno \rightarrow Workdep$ holds in *Emp*, meaning that when *Mgrno* is null, then *Workdept* is also null for the same tuple (only employee No. 20 in example 1).

Example 3. *Assume we are interested in a kind of sequential dependencies [27], i.e. dependencies showing similar behavior of attribute values. Q_3 discovers numerical attributes that vary together (i.e., $X \rightarrow Y$ means that if X increases then Y also increases).*

Q_3:
```
FINDRULES
   OVER Educlevel, Sal, Bonus, Comm
   SCOPE t1, t2 Emp
   CONDITION ON $A IS t1.$A >= t2.$A
```

Using Q_3, *Sal* $\rightarrow$ *Comm* and *Comm* $\rightarrow$ *Sal* hold in *Emp*, which means that a higher salary is equivalent to a higher commission.

Example 4. *Tuple-variables can also be defined on different relations. For instance, the following query searches for inequalities between tuple-variables from two views, referred to as `managers` and `managees` in the query.*

```
FINDRULES OVER Educlevel, Sal, Bonus, Comm
SCOPE
managers (SELECT * FROM Emp WHERE Empno IN (
```

```
    SELECT Mgrno FROM Emp
)),
managees (SELECT * FROM Emp WHERE Empno NOT IN (
    SELECT Mgrno FROM Emp WHERE Mgrno IS NOT NULL
))
CONDITION ON $A IS managers.$A > managees.$A
```

The rule $\emptyset \rightarrow$ Educlevel then holds, meaning that managers always have a higher education levels than non-managers.

The interested reader may find more intricate examples in [18].

1.2. RQL query processing in a nutshell

This section unfolds one of the provided examples to convey the core ideas underlying RQL query processing, for which a general architecture is given in figure 3, section 5. To do so, we consider the following query Q (simplified from query Q'_1 given in Example 1):

```
Q: FINDRULES
   OVER Empno, Job, Sex, Bonus
   SCOPE t1, t2 (
     SELECT * FROM Emp WHERE Educlevel > 16
   )
   CONDITION ON $A IS t1.$A = t2.$A
```

After the usual lexical and syntactic analysis, RQL query processing is composed of two main steps:

1. Computing, against the database, a representation of the set of rules represented by the query. Such a representation is a set of attribute sets called a *base* of the associated *closure system*.

2. Computing a cover of the set of rules from this base.

The first step builds a base with respect to the condition specified in the `CONDITION` clause of the query. The basic idea is to perform the necessary joins and cross products between the relations of the `SCOPE` clause, and to apply a filter to elicit attribute sets satisfying the criteria.

Interestingly, a single SQL query is enough to compute such a base. With Oracle DBMS, conditions are specified using conditional statements (`CASE WHEN`) and then concatenated (`||` operator) to provide a string representation of attribute sets. Note that such operations are simple enough to be present in every DBMS.

For the previous query Q, the RQL query processor generates the following SQL query, whose output is given in Table 1. This result is no more than a binary relation (called a context), well known in the Formal Concept Analysis (FCA) community.

base
-
Sex Bonus
Job
Job Sex Bonus
Empno Job Sex Bonus

Table 1: Base of Q in *Emp*

```
SELECT DISTINCT (
    CASE WHEN t1.Empno = t2.Empno THEN 'Empno ' END ||
    CASE WHEN t1.Job = t2.Job THEN 'Job ' END ||
    CASE WHEN t1.Sex = t2.Sex THEN 'Sex ' END ||
    CASE WHEN t1.Bonus = t2.Bonus THEN 'Bonus ' END
) as base
FROM (SELECT * FROM Emp WHERE Educlevel > 16) t1,
     (SELECT * FROM Emp WHERE Educlevel > 16) t2
```

From such a base, the RQL query processor may achieve the second step – computing a cover of the set of rules satisfied in the database with respect to Q – for which we mainly reused well-known techniques from database and FCA (see sections 1.3 and 2.4). This cover consists of three functional dependencies:

- *Empno* → *Job Sex Bonus*
- *Sex* → *Bonus*
- *Bonus* → *Sex*

As a matter of fact, the base also serves as a starting point to compute the closure of an attribute set and to decide whether or not a given rule is satisfied. These computations are outlined in Section 2.4.

1.3. Related work

Declarative and generic approaches for pattern mining have been studied by many researchers under different angles (databases, constraint programming or theoretical foundations), see for example [33, 40, 7, 24, 15, 31, 44, 47, 46, 48]. A crucial question underlying all these approaches is about the languages to first specify the data of interest and second, to express patterns of interest. A survey of data mining query languages has been done in [12] and the cross-fertilization between data mining and constraint programming has been investigated recently in [31, 44].

Nevertheless, we argue that pattern mining languages should benefit from direct extensions of SQL languages, since data are most of the time stored in DBMSs. Practical approaches, as close as possible to DBMSs, have been proposed for example in [43, 22, 49, 13] to interact more directly with DBMSs query engines. Nevertheless, as far as we know, we are not aware of systems similar to RQL since it targets a very specific class of rules, i.e. those rules verifying Armstrong axioms and thus classical implication in logics. In other words, association rules systems such as MINE RULE [43] or data mining extensions of SQL systems (e.g. Oracle, IBM) cannot be fairly compared to RQL. Moreover,

RQL cannot be compared to association rules systems due to the frequency requirement.

The $SafeRL$ language, inspired from the logical language proposed in [15], goes into this direction by providing a formal semantic based on the tuple relational calculus (TRC), underpinning SQL. FDs, association rules with 100% confidence and the ad-hoc language proposed in [6] are special cases of our $SafeRL$ language but none of them has a logical query language foundation. In this paper, we provide a **logical view** on the ad-hoc language proposed in [6] for gene expression data. The language of [6] was defined using a BNF grammar with no clear logical foundation and hence no practical SQL-like query language. From a technical point of view, the paper can be seen as a generalization of the approach proposed in [39, 20, 38] to discover functional dependencies (FD). Indeed, the machinery introduced in this paper **generalizes** the computation of *agree sets* (a base of the closure system associated with FD) to **every RQL query**. Moreover, the paper is a major extension of [5, 17]: [5] provides a logical view on the ad-hoc language proposed in [6], with some restrictions (e.g. only one relation schema). Theoretical results on decidability problems are also given. We do not reproduce all the results in this paper. [17] introduces the practical query language RQL for the first time. Moreover, RQL has been used as a core building block to build a complete suite for genomics-data analysis in the RulNet prototype [52]. In RulNet, RQL is used with a slightly different syntax but with the same query engine to process the queries. Many options are provided to conveniently display a set of rules as gene regulatory network and to assess the quality of rules with different interestingness measures. Interestingly, other applications such as [3] exist for data analysis in biomedical studies with implications, for which RQL could also be useful.

Many dependencies different from functional dependencies have been studied such as implications in formal concept analysis (FCA) [38, 42, 9], conditional FDs [14], sequential dependencies [27], metric FDs [35], denial constraints [19] or constraint-generating dependencies [10]. They can partially be represented with RQL (cf. examples in Section 1) with some restrictions though. Nevertheless, RQL widens the scope of dependencies and lets the data analysts decide their patterns of interest with respect to their background knowledge, without any presupposition on the dependencies to be discovered.

1.4. 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 $SafeRL$ language, while section 4 presents some results used for computing the answer to $SafeRL$ queries. Section 5 presents experimental results, section 6 presents the web prototype for RQL and section 7 concludes.

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 [1].

Fix a finite universe $\mathbb{U}$ of *attributes* (denoted by $\mathtt{A}, \mathtt{B}, \ldots$), a countably infinite domain $\mathbb{D}$ of *constants* (denoted by $\mathtt{c}, \mathtt{c}', \ldots$) and a finite set $\mathbb{R}$ of *relation symbols* (denoted by $\mathtt{R}, \mathtt{S}, \ldots$). $\mathbb{U}, \mathbb{D}, \mathbb{R}$ are pairwise disjoint. 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 $\mathtt{t} : \mathtt{R} \to \mathbb{D}$. 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. $\mathbf{R} = \{\mathtt{R_1}, \ldots, \mathtt{R_n}\}$. A *database instance* (or simply a database) is a function d from $\mathbf{R}$ to the set of possible relations such that $d(\mathtt{R_i}) = \mathtt{r_i}$, with $\mathtt{r_i}$ 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

Since TRC is a core component of $SafeRL$, we recall here the syntax and semantics of the TRC in its simplest form (see [1] for more details). 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

$$q = \{t \mid \psi\}$$

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 consider *safe TRC*, the fragment of TRC known to always provide finite answers [1].

Moreover, we shall admit several tuple variables in TRC formulas: Let $\psi(t_1, \ldots, t_n)$ be a safe TRC formulas with $t_1, \ldots, t_n$ free variables, i.e. $q = \{(t_1, \ldots, t_n) \mid \psi(t_1, \ldots, t_n)\}$. Then the answer is just a set of tuple assignment σ defined for each $t_i, i = 1..n$, i.e. $ans(q, d) = \{\sigma_{\{t_1, \ldots, t_n\}} \mid \langle d, \sigma \rangle \models \psi(t_1, \ldots, t_n)\}$.

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 applications in databases for functional dependencies [8] and in formal concept analysis for implications [26]. The interested reader should refer to [16, 11] for a comprehensive survey. We summarize the main results that are useful for the rest of the paper.

Let $U \subseteq \mathbb{U}$ and $X, Y \subseteq \mathbb{U}$. A rule is a syntactic expression of the form $X \to Y$. $C \subseteq 2^U$ is a closure system if $U \in C$ and $X, Y \in C \Rightarrow X \cap Y \in C$ [26].

Let F be a set of rules on U. A rule $X \to A$ is implied by F, denoted by $F \vdash X \to A$, if there is a derivation (or a proof) from F using Armstrong's axiom system (reflexivity, augmentation, transitivity) ending to $X \to A$. The closure of a set $X \subseteq U$ is defined by $X_F^+ = \{A \in U \mid F \vdash X \to A\}$. A closure system can be defined for F, noted $CL(F) = \{X \subseteq U | X = X_F^+\}$.

Let $IRR(F)$ be the set of meet-irreducible elements of $CL(F)$, i.e. $X \in IRR(F)$ iff for all $Y, Z \in CL(F), (X = Y \cap Z) \Rightarrow (X = Y$ or $X = Z)$.

The notion of *base* of a closure system is now 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 [26].

It is worth noting that whenever a base has been computed from a given relation r, we can address the following problems:

1. Given a set of attributes, compute its closure with respect to the rules satisfied in r.

2. Given a rule, say whether or not this rule is satisfied. If not, give a counter-example from r.

3. Compute a cover of non satisfied rules in r.

4. Compute a cover (or basis) of satisfied rules in r.

Let us consider the well-known functional dependencies. Given a relation r over R, a base of the closure system associated to satisfied FDs in r is known as the set of *agree sets*. Given the agree sets $ag(r)$, each problem listed above has been studied extensively in database and formal concept analysis communities [38]:

1. Compute the closure of a set of attributes: let $X \subseteq R$. $X^+ = \bigcap\{Y \in ag(r) | X \subseteq Y\}$.

2. Verify a rule: let $X \rightarrow Y$ be a rule, $r \models X \rightarrow Y$ iff $Y \subseteq X^+$.

3. Compute a cover of approximate FDs, known as the Gottlob and Libkin cover [29]: $X \rightarrow A$ is given whenever $X \in max(A, r) = max_{\subseteq}\{X \in ag(r) | A \notin X\}$.

4. Compute the canonical cover[5] for satisfied FDs [39, 20, 38]: $X \rightarrow A$ is given whenever X is a minimal transversal of the hypergraph $\{R \setminus X | X \in max(A, r)\}$.

From a complexity point of view, steps 1 to 3 described above are polynomial while the fourth one is incremental quasi-polynomial in the size of the input and the output due to the enumeration of minimal transversal [25].

The rest of this paper proposes a generalization of this approach. Indeed, each $SafeRL$ query defines a closure system and therefore, in order to reuse previous results, the problem turns out to be on *the computation against the database of a base with respect to a given $SafeRL$ query*. In this paper, we will focus on this problem only. Other problems listed above such as the generation of a canonical cover will not be detailed.

3. A Query Language for Rule Mining

In the introduction, we have illustrated RQL – an SQL-like friendly language – through examples. This section formally defines the syntax and semantics of $SafeRL$ from which RQL is derived. We have introduced safe TRC for expressing SQL-like queries. Before defining $SafeRL$, it remains to precisely define the notion of *mining formulas*, denoted in RQL (cf. previous examples) as:

`CONDITION ON` $\$A$ `IS` $\delta(\$A, t_1, \ldots, t_n)$

[5] Also called *canonical direct basis* [11]

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 $\mathsf{c} \in \mathbb{D}$. A mining formula is of the form: $t_1.A = t_2.B \mid t.A = \mathsf{c} \mid \neg\delta \mid \delta_1 \wedge \delta_2$ where δ, δ_1, δ_2 are mining formulas.*

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*** $\sigma(t_1)(\rho(A)) = \sigma(t_2)(\rho(B))$
- $\langle \sigma, \rho \rangle \models t.A = \mathsf{c}$ ***iff*** $\sigma(t)(\rho(A)) = \mathsf{c}$
- $\langle \sigma, \rho \rangle \models \neg\delta$ ***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 relatively restrictive to keep the presentation simple. In practice, we shall use other binary operators such as $\leq, <, \geq, >, \ldots$ Details are omitted.

3.2. $SafeRL$ queries

The $SafeRL$ query language can now be defined.

Definition 3. *A $SafeRL$ query over a database schema $\mathbf{R}$ is an expression of the form:*

$$Q = \{\, X \to Y \mid \forall t_1 \ldots \forall t_n \, \big[\psi(t_1, \ldots, t_n) \to \\ \big(\forall A \in X(\delta(A, t_1, ..., t_n)) \to \forall A \in Y(\delta(A, t_1, ..., t_n))\big)\big] \,\}$$

where:

- *X and 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 single free attribute-variable.*

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

Example 5. *Continuing example 1, query Q_1 is formalized in $SafeRL$ as follows:*

$$Q_1 = \{\, X \to Y \mid \forall t_1, t_2 \, \big[\big(\mathtt{EMP}(t_1) \wedge \mathtt{EMP}(t_2)\big) \to \\ \big(\forall A \in X(t_1.A = t_2.A) \to \forall A \in Y(t_1.A = t_2.A)\big)\big] \,\}$$

More succinctly, Q_1 is also noted $\langle \mathtt{EMP}(t_1) \wedge \mathtt{EMP}(t_2), (t_1.A = t_2.A)) \rangle$.

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

To specify the result of the evaluation of a $SafeRL$ query against a database, we define the notion of satisfaction.

Definition 4. *A $SafeRL$ 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 (1)
$\langle d, \sigma \rangle \models \psi$:

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 in a relation, we may have n tuples from the database d satisfying ψ (cf (1)); and instead of the condition "for all $A \in X, t_1[A] = t_2[A]$", we have "for all $\mathtt{A} \in \Sigma(X), \delta(\mathtt{A}, t_1, \ldots, t_n)$" (cf. (2) and (3)).

Definition 5. *The answer of a $SafeRL$ query $Q = \langle \psi, \delta \rangle$ in a database d over* $\mathbf{R}$*, denoted by $ans(Q, d)$, is defined as:*

$$ans(Q, d) = \{\Sigma(X) \rightarrow \Sigma(Y) \mid \langle d, \Sigma \rangle \models \langle \psi, \delta \rangle, \Sigma(X) \cup \Sigma(Y) \subseteq sch(Q)\}$$

3.3. RQL: A practical language for $SafeRL$

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

```
FINDRULES
OVER A_1, ..., A_n
SCOPE t_1(SQL_1), ..., t_n(SQL_n)
WHERE condition(t_1, ..., t_n)
CONDITION ON $A IS δ($A, t_1, ..., t_n)
```

The `FINDRULES` clause identifies RQL queries. The `OVER` clause specifies the set of attributes to be used for rule discovery. Those attributes have to be included in $sch(Q)$. The `SCOPE` clause specifies every tuple to be used to discover the rules and corresponds to the tuple-variables of ψ. Each tuple-variable is associated with an SQL query (which can be factorized if they refer to the same SQL query). The `WHERE` clause is borrowed from the SQL `WHERE` clause to specify relationships between tuple variables. The `CONDITION ON` $\$A$ `IS` specifies the mining formula δ.

Note that RQL allows much more flexibility than $SafeRL$ since more advanced conditions available in SQL – such as regular expressions, user-defined functions or built-in functions of the underlying DBMS – can be used for free, as in query Q_1'' of Example 1.

4. Theoretical results

In [5], a slightly different language for rule mining has been proposed. One of the main results was to point out that every query was "Armstrong-compliant", meaning basically that Armstrong axioms are sound and that each query defines a closure system. The same result holds for $SafeRL$ queries as we shall see. This result means that for query processing, we can adopt a strategy that generalizes the process of FD inference through agree sets [39, 37].

Given a database d and a $SafeRL$ 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, then the notion of base.

4.1. Closure system and bases for $SafeRL$ 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 $ans(Q, d)$.

Definition 6. *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:* $CL_Q(d) = \{\mathtt{Z} \subseteq sch(Q) \mid \mathtt{Z} \text{ satisfies } ans(Q, d)\}$.

Lemma 1. *Let* Q *be a* $SafeRL$ *query and* d *a database.*
Then, $CL_Q(d)$ *is a closure system on* $sch(Q)$.

Proof 1. *We have to show that* $sch(Q) \in CL_Q(d)$ *and for all* $\mathtt{X}, \mathtt{Y} \in CL_Q(d)$, $\mathtt{X} \cap \mathtt{Y} \in CL_Q(d)$.

- $sch(Q)$ *satisfies* $ans(Q, d)$ *since for every* $\mathtt{X} \rightarrow \mathtt{Y} \in ans(Q, d)$, *we have* $\mathtt{Y} \subseteq sch(Q)$.
- *Let* $\mathtt{X}, \mathtt{Y} \in CL_Q(d)$. *We have to show that* $\mathtt{X} \cap \mathtt{Y} \in CL_Q(d)$, *i.e.* $\mathtt{X} \cap \mathtt{Y}$ *satisfies* $ans(Q, d)$. *By hypothesis, we have* $\mathtt{X} \in CL_Q(d)$ *and* $\mathtt{Y} \in CL_Q(d)$, *implying for every* $\mathtt{V} \rightarrow \mathtt{W} \in ans(Q, d)$, *(*$\mathtt{V} \not\subseteq \mathtt{X}$ *or* $\mathtt{W} \subseteq \mathtt{X}$*) and (*$\mathtt{V} \not\subseteq \mathtt{Y}$ *or* $\mathtt{W} \subseteq \mathtt{Y}$*). We deduce that* $\mathtt{V} \not\subseteq \mathtt{X} \cap \mathtt{Y}$ *or* $\mathtt{W} \subseteq \mathtt{X} \cap \mathtt{Y}$ *and the result follows.*

Corollary 1. *Let* Q *be a* $SafeRL$ *query and* d *a database.*
Then, Q *and* d *define a unique closure system.*

It turns out that for every RQL query, there exists a closure system and thus, the main problem is to effectively compute a representation of the closure system, i.e. a base (cf. definition 1).

In our setting, the definition of the base is:

Definition 7. *Let* $Q = \langle \psi, \delta \rangle$ *be a* $SafeRL$ *query over* $\mathbf{R}$ *and* d *a database over* $\mathbf{R}$. *We assume that the attribute variable in* δ *is* A. *The* base *of* Q *in* d, *denoted by* $B_Q(d)$, *is defined by:*

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

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 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 $CL_Q(d)$.*

Proof 2. *We have to show that $B_Q(d) \subseteq CL_Q(d)$ and $IRR(CL_Q(d)) \subseteq B_Q(d)$.*

Let $\mathtt{Z} \in B_Q(d)$ i.e. there exists a set of tuples $\mathtt{t_1}$ over $sch(t_1)$, ..., $\mathtt{t_n}$ over $sch(t_n)$ with $\langle d, \sigma_{t_1\mapsto \mathtt{t_1},\ldots,t_n\mapsto \mathtt{t_n}}\rangle \models \psi(t_1,\ldots,t_n)$ such that:

- *$\forall \mathtt{A} \in \mathtt{Z}, \langle \sigma_{t_1\mapsto \mathtt{t_1},\ldots,t_n\mapsto \mathtt{t_n}}, \rho_{A\mapsto \mathtt{A}}\rangle \models \delta(A, t_1,\ldots,t_n)$, and*
- *$\forall \mathtt{A} \in sch(Q)\backslash \mathtt{Z}, \langle \sigma_{t_1\mapsto \mathtt{t_1},\ldots,t_n\mapsto \mathtt{t_n}}, \rho_{A\mapsto \mathtt{A}}\rangle \not\models \delta(A, t_1,\ldots,t_n)$.*

We have to show that $\mathtt{Z} \in CL_Q(d)$. Suppose $\mathtt{Z} \notin CL_Q(d)$. In that case, there exists $\mathtt{X} \subseteq \mathtt{Z}$ and $\mathtt{Y} \not\subseteq \mathtt{Z}$ such that $\mathtt{X} \to \mathtt{Y} \in ans(Q,d)$. It implies that: $\forall \mathtt{A} \in \mathtt{Y}$, $\langle \sigma_{t_1\mapsto \mathtt{t_1},\ldots,t_n\mapsto \mathtt{t_n}}, \rho_{A\mapsto \mathtt{A}}\rangle \models \delta(A, t_1,\ldots,t_n)$ since $\mathtt{X} \subseteq \mathtt{Z}$, there is a contradiction since $\mathtt{Y} \not\subseteq \mathtt{Z}$.

Let $\mathtt{Z} \in IRR(CL_Q(d))$, we have to show that $\mathtt{Z} \in B_Q(d)$. We know that $\mathtt{Z} \in CL_Q(d)$. Then, we have for all $\mathtt{A} \in sch(Q)\backslash \mathtt{Z}$, $\mathtt{Z} \to \mathtt{A} \notin ans(Q,d)$. This means there exists some $\sigma_{\mathtt{A}}$ such that $\langle \sigma_{\mathtt{A}}, \rho_{A\mapsto \mathtt{A}}\rangle \not\models \delta(A, t_1,\ldots,t_n)$ and for all $\mathtt{B} \in \mathtt{Z}$, $\langle \sigma_{\mathtt{A}}, \rho_{A\mapsto \mathtt{B}}\rangle \models \delta(A, t_1,\ldots,t_n)$. In other words, for all $\mathtt{A} \in sch(Q)\backslash \mathtt{Z}$, there exists $\mathtt{X} \in B_Q(d)$ such that $\mathtt{Z} \subseteq \mathtt{X}$ and $\mathtt{A} \notin \mathtt{X}$. Since $\mathtt{Z} = \bigcap_{\mathtt{A}\in sch(Q)\backslash \mathtt{Z}}\{\mathtt{X} \in B_Q(d) | \mathtt{Z} \subseteq \mathtt{X}$ and $\mathtt{A} \notin \mathtt{X}\}$ and $\mathtt{Z} \in IRR(CL_Q(d))$, we have the result, i.e. $\mathtt{Z} \in B_Q(d)$.

4.2. Computing the base of a $SafeRL$ query using query rewriting

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

For every 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 single SQL query, i.e. the computation of 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 have to transform $Q = \langle \psi(t_1,\ldots,t_n), \delta(A, t_1,\ldots,t_n)\rangle$ into $Q' = \langle \psi'(t), \delta'(A,t)\rangle$.

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 as follows: $\mathtt{R} = \bigcup_{i\in 1..n}\{\langle \mathtt{A}, i\rangle \mid \mathtt{A} \in sch(t_i)\}$. Let t be a fresh tuple variable with $sch(t) = \mathtt{R}$ and $A_1,\ldots,A_n$ be n fresh attribute variables.

Example 6. *Let us consider the relation Emp over attributes Empno, Lastname, Workdept shortened in the following examples to employees No. 10, 90 and 100. Let $Q_1 = \langle \mathtt{Emp}(t_1) \wedge \mathtt{Emp}(t_2), (t_1.A = t_2.A)\rangle$.*

Emp	*Empno*	*Lastname*	*Workdept*
$\mathtt{t}_1$	*10*	*SPEN*	*C01*
$\mathtt{t}_2$	*90*	*HENDER*	*D21*
$\mathtt{t}_3$	*100*	*SPEN*	*C01*

Given Q_1, its rewriting is defined over the following new schema Emp^*: $\{\langle Empno,\ 1\rangle, \langle Lastname,\ 1\rangle, \langle Workdept,\ 1\rangle, \langle Empno,\ 2\rangle, \langle Lastname,\ 2\rangle, \langle Workdept,\ 2\rangle\}$. The rewriting will use a fresh variable $t : Emp^*$ and two fresh attribute variables A_1 and A_2.

Now, the rewriting function rw has to be defined for TRC formulas, mining formulas and has to overload tuple variable assignments and attribute variable assignments.

For mining formulas, the rewriting is defined inductively. Let δ a mining formulae.

- $\mathsf{rw}(t_i.A = \mathtt{c}) = (t.A_i = \mathtt{c})$
- $\mathsf{rw}(t_i.A = t_j.A) = (t.A_i = t.A_j)$
- $\mathsf{rw}(\neg\delta) = \neg\mathsf{rw}(\delta)$
- $\mathsf{rw}(\delta_1 \wedge \delta_2) = \mathsf{rw}(\delta_1) \wedge \mathsf{rw}(\delta_2)$

Clearly, attribute variable names matter in this rewriting: A_i carries both the name of the initial variable A and the index i of the initial tuple variable.

Example 7. *In Q_1, let us consider the mining formula $\delta = (t_1.A = t_2.A)$. Then $\mathsf{rw}(\delta) = (t.A_1 = t.A_2)$.*

We now have to overload rw to transform attribute-assignment ρ and tuple-assignment σ in order to take into account the new fresh variables.

- For each A_i introduced above, we define $\mathsf{rw}(\rho)(A_i) = \langle\rho(A), i\rangle$, i.e. the name of the variable A_i allows to identify both the attribute variable A and the i^{th} part of the tuple on the new schema $\mathtt{R}$.
- Let $t : \mathtt{R}$ the tuple variable introduced above. We define $\mathsf{rw}(\sigma)(t) = \mathtt{t}$ such that $\mathtt{t}(\langle\mathtt{A}, i\rangle) = \sigma(t_i)(\mathtt{A})$ for each $\langle\mathtt{A}, i\rangle \in \mathtt{R}$. Moreover, the tuple assigned to each t_i by σ is kept in $\mathsf{rw}(\sigma)$: for $i \in \{1, \ldots, n\}$, $\mathsf{rw}(\sigma)(t_i) = \sigma(t_i)$.

Example 8. *Continuing the previous example with the following attribute-assignment $\rho_0(A) = Lastname$. We have:*

- $\mathsf{rw}(\rho_0)(A_1) = \langle\rho_0(A), 1\rangle = \langle Lastname, 1\rangle$
- $\mathsf{rw}(\rho_0)(A_2) = \langle\rho_0(A), 2\rangle = \langle Lastname, 2\rangle$

Then, with the following tuple-assignment $\sigma_0(t_1) = \mathtt{t}_1$, $\sigma_0(t_2) = \mathtt{t}_2$, we have:

- $\mathsf{rw}(\sigma_0)(t) = \mathtt{t}$ *such that* $\forall\mathtt{A} \in \{Empno,\ Lastname,\ Workdept\}$, $\mathtt{t}(\langle\mathtt{A}, 1\rangle) = \sigma_0(t_1)(\mathtt{A}) = \mathtt{t}_1(\mathtt{A})$ *and* $\mathtt{t}(\langle\mathtt{A}, 2\rangle) = \sigma_0(t_2)(\mathtt{A}) = \mathtt{t}_2(\mathtt{A})$.

In other words, the tuple $\mathtt{t}$ *defined over Emp* is equal to the concatenation of* $\mathtt{t_1}$ *and* $\mathtt{t_2}$ *in the original relation, namely* $\langle \mathtt{10}, \mathtt{SPEN}, \mathtt{C01}, \mathtt{90}, \mathtt{HENDER}, \mathtt{D21} \rangle$ *(cf. Figure 2 where* $\mathtt{t}$ *is represented by* $\mathtt{t'_1}$*).*

Given a mining formula δ, we have the following lemma that gives an equivalence with the previous rewriting $\mathsf{rw}(\delta)$.

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

Proof 3. *By induction on* δ*:*

- *If* $\delta = \neg\delta'$*:* $\mathsf{rw}(\delta) = \neg\mathsf{rw}(\delta')$*. By induction,* $\langle \sigma, \rho \rangle \not\models \delta'$ *iff* $\langle \mathsf{rw}(\sigma), \mathsf{rw}(\rho) \rangle \not\models \mathsf{rw}(\delta')$*. Therefore* $\langle \sigma, \rho \rangle \models \delta$ *iff* $\langle \mathsf{rw}(\sigma), \mathsf{rw}(\rho) \rangle \models \mathsf{rw}(\delta)$*.*
- *If* $\delta = \delta_1 \wedge \delta_2$*:* $\mathsf{rw}(\delta) = \mathsf{rw}(\delta_1) \wedge \mathsf{rw}(\delta_2)$*. By induction,* $\langle \sigma, \rho \rangle \models \delta_1$ *iff* $\langle \mathsf{rw}(\sigma), \mathsf{rw}(\rho) \rangle \models \mathsf{rw}(\delta_1)$ *and* $\langle \sigma, \rho \rangle \models \delta_2$ *iff* $\langle \mathsf{rw}(\sigma), \mathsf{rw}(\rho) \rangle \models \mathsf{rw}(\delta_2)$*. Therefore* $\langle \sigma, \rho \rangle \models \delta_1 \wedge \delta_2$ *iff* $\langle \mathsf{rw}(\sigma), \mathsf{rw}(\rho) \rangle \models \mathsf{rw}(\delta_1) \wedge \mathsf{rw}(\delta_2)$*.*
- *If* $\delta = (t_i.A = t_j.A)$*:* $\mathsf{rw}(\delta) = (t.A_i = t.A_j)$*. Let* $\sigma' = \mathsf{rw}(\sigma)$ *and* $\rho' = \mathsf{rw}(\rho)$*. By definition of* rw*:* $\sigma(t_i)(\rho(A)) = \sigma'(t)(\langle \rho(A), i \rangle) = \sigma'(t)(\rho'(A_i))$ *and* $\sigma(t_j)(\rho(A)) = \sigma'(t)(\rho'(A_j))$*.*

  Let us assume that $\langle \sigma, \rho \rangle \models \delta$*. Then* $\sigma(t_i)(\rho(A)) = \sigma(t_j)(\rho(A))$*. Therefore* $\sigma'(t)(\rho'(A_i)) = \sigma'(t)(\rho'(A_j))$ *and* $\langle \sigma', \rho' \rangle \models t.A_i = t.A_j$*.*

  Conversely if $\langle \sigma', \rho' \rangle \models \mathsf{rw}(\delta)$*, then* $\sigma'(t)(\rho'(A_i)) = \sigma'(t)(\rho'(A_j))$*. Therefore* $\sigma(t_i)(\rho(A)) = \sigma(t_j)(\rho(A))$ *and* $\langle \sigma, \rho \rangle \models t_i.A = t_j.A$*.*
- *The case where* $\delta = (t_i.A = \mathsf{c})$ *is similar to the previous one.*

Finally, it remains to rewrite the TRC formula $\psi(t_1, \ldots, t_n)$ with $\mathsf{rw}(\psi)(t)$ by forcing t to have each t_i as one of its components:

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

Example 9. *Continuing the previous example, we have* $\psi(t_1, t_2) = Emp(t_1) \wedge Emp(t_2)$ *and* $sch(Q_1) = \{Empno, Lastname, Workdept\}$*.*
Then $\mathsf{rw}(\psi) = \exists\, t_1 : sch(t_1)(\exists\, t_2 : sch(t_2)(\zeta))$ *such that* ζ *is the conjunction of the following conditions:*

- $Emp(t_1)$
- $Emp(t_2)$
- $t.\langle Empno, 1 \rangle = t_1.Empno$
- $t.\langle Empno, 2 \rangle = t_2.Empno$
- $t.\langle Lastname, 1 \rangle = t_1.Lastname$

$\texttt{Emp}^*$	$\langle Empno, 1\rangle$	$\langle Lastname, 1\rangle$	$\langle Workdept, 1\rangle$	$\langle Empno, 2\rangle$	$\langle Lastname, 2\rangle$	$\langle Workdept, 2\rangle$
$\texttt{t}'_1$	10	SPEN	C01	90	HENDER	D21
$\texttt{t}'_2$	10	SPEN	C01	100	SPEN	C01
$\texttt{t}'_3$	90	HENDER	D21	100	SPEN	C01

Figure 2: Query rewriting for *Emp*

- $t.\langle Lastname, 2\rangle = t_2.Lastname$
- $t.\langle Workdept, 1\rangle = t_1.Workdept$
- $t.\langle Workdept, 2\rangle = t_2.Workdept$

Given a safe TRC formula $\psi(t_1, \ldots, t_n)$, we have the following result that gives an equivalence with the previous rewriting $\mathsf{rw}(\psi)(t)$.

Lemma 3. $\langle d, \sigma\rangle \models \psi$ *iff* $\langle d, \mathsf{rw}(\sigma)\rangle \models \mathsf{rw}(\psi)$

Proof 4. *Let us assume that* $\langle d, \sigma\rangle \models \psi$.

Since t *is not a tuple-variable of* ψ, $\langle d, \mathsf{rw}(\sigma)\rangle \models \psi$. *By definition of* $\mathsf{rw}(\sigma)$, $\langle d, \mathsf{rw}(\sigma)\rangle \models \bigwedge_{\texttt{A}\in sch(Q)} \bigwedge_{i=1}^{n} t.\langle \texttt{A}, i\rangle = t_i.\texttt{A}$. *Therefore* $\langle d, \mathsf{rw}(\sigma)\rangle \models \exists t_1(\ldots(\exists t_n(\psi \wedge \bigwedge_{\texttt{A}\in sch(Q)} \bigwedge_{i=1}^{n} t.\langle \texttt{A}, i\rangle = t_i.\texttt{A}))\ldots)$, *i.e.* $\langle d, \mathsf{rw}(\sigma)\rangle \models \mathsf{rw}(\psi)$.

Let us now assume that $\langle d, \mathsf{rw}(\sigma)\rangle \models \mathsf{rw}(\psi)$ *and consider the following formula:* $\psi(t_1, \ldots, t_n) \wedge \bigwedge_{\texttt{A}\in sch(Q)} \bigwedge_{i=1}^{n} t.\langle \texttt{A}, i\rangle = t_i.\texttt{A}$. *Then there exists* $\texttt{t}_1, \ldots, \texttt{t}_\texttt{n}$ *and* σ' *such that* $\sigma' = \mathsf{rw}(\sigma)_{t_1 \mapsto \texttt{t}_1, \ldots, t_n \mapsto \texttt{t}_\texttt{n}}$ *and* $\langle d, \sigma'\rangle \models \psi \wedge \bigwedge_{\texttt{A}\in sch(Q)} \bigwedge_{i=1}^{n} t.\langle \texttt{A}, i\rangle = t_i.\texttt{A}$. *For* $\langle d, \sigma'\rangle \models \bigwedge_{\texttt{A}\in sch(Q)} \bigwedge_{i=1}^{n} t.\langle \texttt{A}, i\rangle = t_i.\texttt{A}$ *to hold, we necessarily have* $\texttt{t}_\texttt{i} = \sigma(t_i)$ *for* $1 \leq i \leq n$. *Since* $\langle d, \sigma'\rangle \models \psi$, *we deduce* $\langle d, \sigma\rangle \models \psi$.

Example 10. *Continuing the previous example, let us consider* $\psi(t_1, t_2)$, *its rewriting* $\mathsf{rw}(\psi)(t)$, *the associated query* $\{t \mid \mathsf{rw}(\psi)(t)\}$ *and its evaluation against Emp. Figure 2 gives the result of this rewritten query. Note that, to avoid considering each pair of employees twice, we have restricted this query to* $t1.Empno < t2.Empno$ *i.e.* $Q_1 = \langle \texttt{Emp}(t_1) \wedge \texttt{Emp}(t_2) \wedge t1.Empno < t2.Empno, (t_1.A = t_2.A)\rangle$.

Interestingly, for every $SafeRL$ query $Q = \langle \psi(t_1, \ldots, t_n), \delta((t_1, \ldots, t_n, A)\rangle$, $\mathsf{rw}(\psi)$ gives an intermediate representation allowing to compute the mining formula $\delta(t_1, \ldots, t_n, A)$ on each tuple of the answer set.

Example 11. *Let us consider the second tuple, denoted by* $\texttt{t}'_2$, *given in example 10. The attribute Lastname (resp. Workdept) satisfies* $\mathsf{rw}(\delta)$ *since* $\texttt{t}'_2(\langle Lastname, 1\rangle) = \texttt{t}'_2(\langle Lastname, 2\rangle)$ *(resp.* $\texttt{t}'_2(\langle Workdept, 1\rangle) = \texttt{t}'_2(\langle Workdept, 2\rangle)$*). The attribute Empno does not since* $\texttt{t}'_2(\langle Empno, 1\rangle) \neq \texttt{t}'_2(\langle Empno, 2\rangle)$. *From* $\texttt{t}'_2$, *the maximal set of attributes verifying* $\mathsf{rw}(\delta)$ *is* $\{Lastname, Workdept\}$, *which is a closed set associated to the initial query* Q_1. *The closed set associated with* $\texttt{t}'_1$ *(resp* $\texttt{t}'_3$*) is empty since* $\forall \texttt{A} \in sch(Q_1)$, $\texttt{t}'_1(\langle \texttt{A}, 1\rangle) \neq \texttt{t}'_1(\langle \texttt{A}, 2\rangle)$ *(resp.* $\texttt{t}'_3(\langle \texttt{A}, 1\rangle) \neq \texttt{t}'_3(\langle \texttt{A}, 2\rangle)$*).*

We can now give an important result of the paper, i.e. how a base of the closure system associated to every $SafeRL$ query can be computed.

Proposition 2.

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

Proof 5. *By definition:*

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

Since A is the only attribute-variable in δ:

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

By lemma 2:

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

By lemma 3:

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

We conclude by remarking that rw *is bijective for* σ*.*

The main theorem can now be given.

Theorem 1. *Let Q be a $SafeRC$ query and d a database.*
Then

1. *$ans(Q,d)$ defines a unique closure system on $sch(Q)$, denoted by $CL(ans(Q,d))$.*
2. *There exists an SQL query Q' over d such that $IRR(ans(Q,d)) \subseteq ans(Q',d) \subseteq CL(ans(Q,d))$.*

Proof 6. *The first item follows from Lemma 1. The second one follows from Proposition 1 and 2 for safe TRC formula. Since SQL is at least as expressive as safe TRC, the result holds.*

Therefore $B_Q(d)$ is computable by running only one SQL query, corresponding exactly to the safe TRC query $\{t \mid \mathsf{rw}(\psi)\}$ with only one difference: the `SELECT` part has to evaluate the satisfaction of $\mathsf{rw}(\delta)$. From a practical point of view, it means that the base can be computed with only one SQL statement for all $SafeRC$ queries, pushing query processing as much as possible into the DBMS.

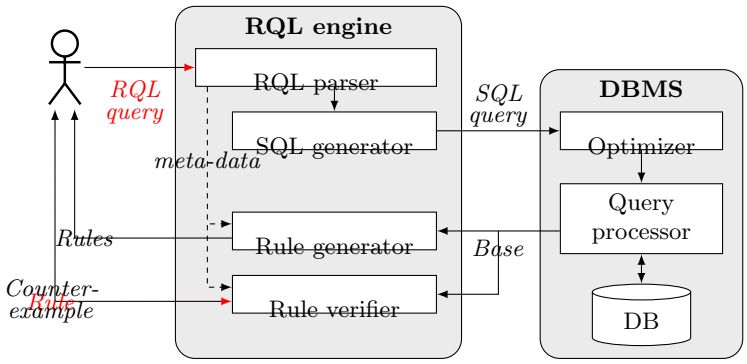


Figure 3: RQL query processing overview

5. Implementation and experiments

Given a 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. From the base, compute the canonical cover for exact rules and a Gottlob and Libkin cover for approximate rules [29]. Details are out of the scope of this paper, we mainly reused the code of T. Uno [45] for the most expensive part of the rule generation process, i.e. the enumeration of minimal transversal of hypergraphs.

Figure 3 gives an overview of RQL query processing.

5.1. Sensor Data

We experimented our RQL processing engine using the PlaceLab dataset provided by the *MIT House_n Consortium* and *Microsoft Research* [34].

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 4.

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)

Figure 4: 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

Figure 5: Sensors data after SQL preprocessing

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 5, has 165 attributes and 32543 tuples. Apart from the time, each attribute is associated either with one of the 106 physical properties sensors or one of the 58 selected switches.

5.2. Experimental Results

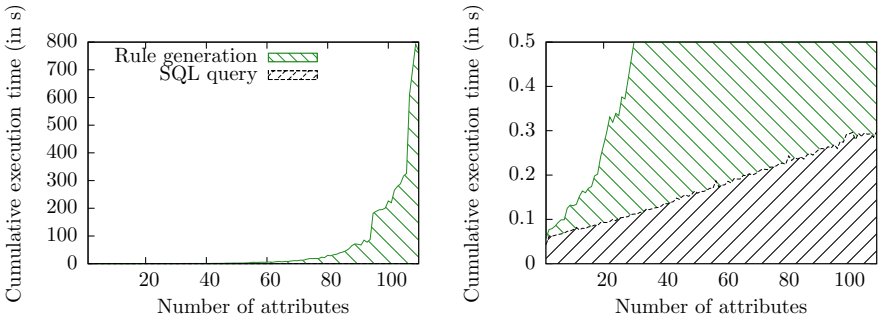
The server used for these experiments is a virtual machine running on VMware 4.1 with 2× Intel Xeon X5650 and 7.2K disks in RAID 5. This virtual machine, running Debian 6.0, disposes of 16 GB of RAM and 4 CPU cores. The DBMS is Oracle Database 11g Release 2.

In these experiments, we consider three families of RQL queries.

Null values (Q_3). This first set of queries mine rules between sensors for null values. Such queries can be used to help identify groups of sensors which are unavailable simultaneously, due, for example, to a shared acquisition system.

Q_3:
```
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 ($A \rightarrow F$, $F \rightarrow C$, $BC \rightarrow E$, $BE \rightarrow D$, $CD \rightarrow E$, $CE \rightarrow A$). Note that, following Armstrong's axioms, rules such as $BC \rightarrow ADEF$ can be inferred. Using SQL queries, we can then confirm that these sensors are indeed almost always unavailable simultaneously. Similarly, we can group four sensors from the living



(a) Rule generation

(b) Zoom on SQL queries durations

Figure 6: Execution time for null rules

room, two sensors from the hall with three sensors from the kitchen, etc. This information could especially be useful if we had lost metadata on sensors locations and descriptions.

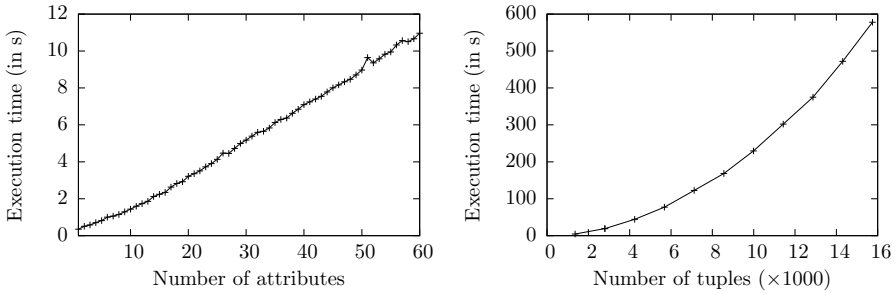
Figure 6 gives the cumulative execution time for various number of attributes in the *OVER* clause of Q1. As expected, rule generation is by far the most expensive part when the query runs over a large set of attributes. The SQL query however lasts less than a second and increases linearly: computation of the base by the DBMS is efficient.

For the subsequent experiments, we focus on the execution time of the base computation. As seen in section 2.4, the data analyst does not always need to generate a canonical cover of the rules, and can address other problems directly from the base.

Functional dependencies (Q_4). This second set of queries finds FDs within a time window. This time window is specified using SQL conditions on the timestamp.

Q_4:
```
FINDRULES
  OVER <list of attributes>
  SCOPE t1, t2 (
    SELECT * FROM sensors
    WHERE time BETWEEN '2006-08-22 00:00:00'
                   AND <end date>
  )
  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 RQL queries. Figure 7 shows the scalability of the base generation with respect to the number of attributes (#tuples = 1439) and the number of tuples (#attributes = 37). As expected, the execution time



(a) Scalability w.r.t. #attributes (b) Scalability w.r.t. #tuples

Figure 7: Execution time for functional dependencies

increases quadratically with the number of tuples due to the theta self-join.

Local maximums (Q_5). 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. $X \rightarrow A$ 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).

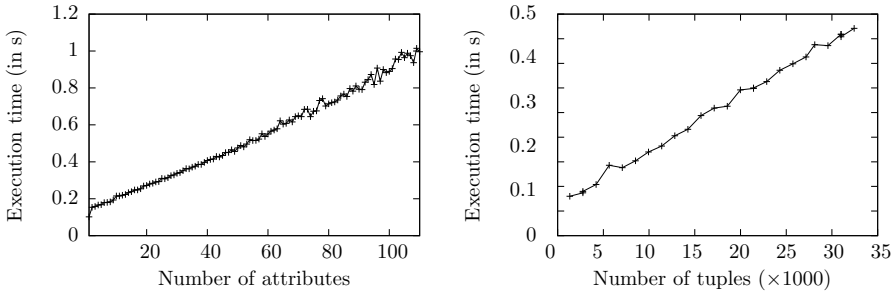
Q_5 :
```
FINDRULES
  OVER <list of attributes>
  SCOPE t1, t2, t3 sensors
  WHERE t2.time = t1.time + interval '1' minute
    AND t3.time = t2.time + interval '1' minute
  CONDITION ON $A IS t1.$A < t2.$A AND t2.$A > t3.$A
```

Even though this query has three tuple-variables, all three are bound by a 1:1 relationship. Consequently, the DBMS computes equi-joins instead of Cartesian products, which improves its efficiency. Figure 8 shows that, similar to Q_3, the execution time grows linearly with both the number of attributes (#tuples = 32433) and the number of tuples (#attributes = 37) – a time window is added to the `SCOPE` clause to adjust the number of tuples, cf. Q_4.

5.3. Comparison with functional dependencies miners

Due to its general application case, RQL can be compared with existing tools for several data mining tasks. In this section, we focus on functional dependencies to evaluate how well RQL behaves with respect to dedicated solutions.

We compared RQL with two functional dependency discovery algorithms: TANE [32] and FDEP [23], identified to be among the fastest existing algorithms in a recent survey [51]. The implementations used as a basis for this comparison are taken from the Metanome project [50], along with five datasets: *ncvoter*, *fdreduced30*, *plista*, *flight* and *uniprot*. Other datasets used in this experiment are provided by the UCI machine learning repository [36]. Table 2 gives an overview of these datasets and the corresponding execution times for RQL,



(a) Scalability w.r.t. #attributes (b) Scalability w.r.t. #tuples

Figure 8: Execution time for local maximums rules

FDEP and TANE, which includes: data retrieval from the Oracle database, FD discovery and writing the cover of minimal FDs to an output file.

The execution strategy of RQL is similar to FDEP, where a negative cover of maximal non-FDs is computed first. During this step, both algorithms have to compare records pair-wise, which is integrated in FDEP and delegated to the DBMS by RQL. Their behaviors are therefore comparable, with a good scalability on the number of columns but limited on the number of rows.

TANE implements another discovery strategy based on a bottom-up traversal of a lattice representing all possible subsets of attributes, along with a compact representation of the data (called *partitions*) used to test functional dependencies. In this experiment, TANE is the most efficient strategy on datasets with many rows, but exceeds the memory limit when its lattice becomes too large [51].

Despite not being dedicated to FDs discovery, RQL provides, in some cases (e.g. when the number of generated FDs is large), the most efficient strategy.

6. A web prototype for RQL

A web prototype has been developed and is freely accessible at `http://rql.insa-lyon.fr` for research and educational purposes. Figure 9 gives a preview of this web interface, which provides an unified access to the user's data and pattern mining techniques using two declarative languages: SQL and RQL.

To discover the RQL language, a pre-filled database is made available in a *Sample* mode, with a few selected example written in both SQL and RQL. Data analysts can also upload their own data (up to 200 KB) in a *Sandbox* mode and write their own SQL and RQL queries to discover meaningful rules.

For example a data analyst wants to explore functional dependencies using Q_1. The analyst could generate a canonical cover for FDs, but instead she only wants to investigate if an employee can be identified by its name and department (*Lastname*, *Workdept* $\rightarrow$ *Empno*). RQL answers whether or not this rule holds,

Dataset	Columns (#)	Rows (#)	Size (KB)	FDs (#)	RQL	FDEP	TANE
iris	5	150	4	4	**<0.1**	0.1	0.3
balance-scale	5	625	6	1	**0.1**	0.2	0.4
chess	7	28,056	519	1	214.6	71.6	**1.0**
abalone	9	4,177	187	137	5.9	3.7	**2.3**
nursery	9	12,960	1,036	1	60.9	25.1	**6.4**
breast-cancer	11	699	19	46	**0.4**	0.5	0.9
bridges	13	108	6	142	0.2	**0.1**	0.6
echocardiogram	13	132	6	536	**0.1**	0.2	0.5
adult	15	48,842	3,881	66	1661.7	784.3	**127.1**
letter	17	20,000	696	61	327.7	**158.0**	*ML*
ncvoter	19	1,000	151	758	1.1	**1.0**	1.7
hepatitis	20	155	7	8,250	0.6	**0.5**	5.2
horse	28	368	28	156,723	**2.9**	6.1	*ML*
fdreduced30	30	250,000	69,580	89,571	*TL*	*ML*	**132.9**
plista	63	1,001	575	178,152	**11.2**	16.4	*ML*
flight	109	1,000	569	982,631	**9.3**	139.3	*ML*
uniprot	223	1,000	2439	unknown	*TL*	*ML*	*ML*

TL: time limit of 4 hours exceeded **ML**: memory limit of 10 GB exceeded

Table 2: Execution times in seconds for functional dependency discovery

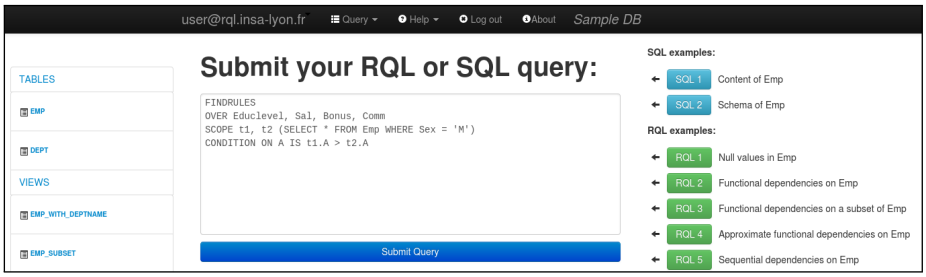


Figure 9: Main web interface

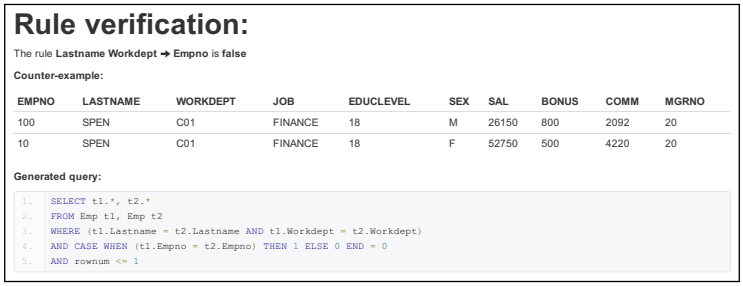


Figure 10: Counter-examples with RQL

and if not, a counter example from the database is displayed. Figure 10 gives an overview of what RQL provides as a counterexample for this rule.

Importantly, the web interface provides the SQL code generated by the system as often as possible, for instance to identify the counterexample, so that the analyst can use this query as a starting point for data exploration [18].

7. Conclusion

In this paper, we have introduced $SafeRQL$, a logical query language based on the tuple relational calculus 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. RQL, the concrete language of $SafeRQL$, is an SQL-like rule 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*. These experiments show both the feasibility and the efficiency of the proposed language.

RQL is an important contribution toward declarative pattern mining whose ambition is to simplify the way data analysts interact with their data. Through a convenient query language, close to SQL, they can focus on scientific discovery instead of spending considerable time switching data between different systems.

As future work, we could generate other covers for rules with RQL, typically optimal covers, namely the Duquenne and Guigues basis [30] or ordered direct implicational basis [4] (see [2] for a recent survey). The scalability of RQL to Big Data remains a quite challenging issue. Even if there is still room for improvements, we believe that alternative approaches should be conducted to guide the search towards interesting rules without enumerating all possible satisfied rules, for example by taking advantage of the knowledge brought by counter-examples. From an application point of view, the potential of RQL could also be studied to express temporal dependencies [53] and more generally for data cleaning [28, 21].

Acknowledegment. This work has been partially funded by the French national research agency (DAG project, ANR-09-EMER, 2009-2013) and CNRS Mastodons projects (PETASKY 2012-2015 and QualiSky 2016).

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] Kira V. Adaricheva and James B. Nation. On implicational bases of closure systems with unique critical sets. *Discrete Applied Mathematics*, 162:51–69, 2014.

[3] Kira V. Adaricheva, James B. Nation, Gordon Okimoto, Vyacheslav Adarichev, Adina Amanbekkyzy, Shuchismita Sarkar, Alibek Sailanbayev, Nazar Seidalin, and Kenneth Alibek. Measuring the implications of the d-basis in analysis of data in biomedical studies. In *Formal Concept Analysis - 13th International Conference, ICFCA 2015, Nerja, Spain, June 23-26, 2015, Proceedings*, pages 39–57, 2015.

[4] Kira V. Adaricheva, James B. Nation, and Robert Rand. Ordered direct implicational basis of a finite closure system. *Discrete Applied Mathematics*, 161(6):707–723, 2013.

[5] Marie Agier, Christine Froidevaux, Jean-Marc Petit, Yoan Renaud, and Jef Wijsen. On Armstrong-compliant Logical Query Languages. In George H. L. Fletcher and Slawek Staworko, editors, *4th International Workshop on Logic in Databases (LID 2011) in conjunction with EDBT/ICDT conference*, pages 33–40. ACM, March 2011.

[6] Marie Agier, Jean-Marc Petit, and Einoshin Suzuki. Unifying framework for rule semantics: Application to gene expression data. *Fundam. Inform.*, 78(4):543–559, 2007.

[7] Hiroki Arimura and Takeaki Uno. Polynomial-delay and polynomial-space algorithms for mining closed sequences, graphs, and pictures in accessible set systems. In *SDM*, pages 1088–1099, 2009.

[8] William Ward Armstrong. Dependency structures of data base relationships. In *Proceedings of the IFIP Congress*, pages 580–583, 1974.

[9] Mikhail A. Babin and Sergei O. Kuznetsov. Computing premises of a minimal cover of functional dependencies is intractable. *Discrete Applied Mathematics*, 161(6):742–749, 2013.

[10] Marianne Baudinet, Jan Chomicki, and Pierre Wolper. Constraint-generating dependencies. *J. Comput. Syst. Sci.*, 59(1):94–115, 1999.

[11] Karell Bertet and Bernard Monjardet. The multiple facets of the canonical direct unit implicational basis. *Theor. Comput. Sci.*, 411(22-24):2155–2166, 2010.

[12] Hendrik Blockeel, Toon Calders, Elisa Fromont, Bart Goethals, Adriana Prado, , and Céline Robardet. A practical comparative study of data mining query languages. In Sašo Džeroski, Bart Goethals, and Panče" Panov, editors, *Inductive Databases and Constraint-Based Data Mining*, pages 59–77. Springer New York, 2010.

[13] Hendrik Blockeel, Toon Calders, Élisa Fromont, Bart Goethals, Adriana Prado, and Céline Robardet. An inductive database system based on virtual mining views. *Data Min. Knowl. Discov.*, 24(1):247–287, 2012.

[14] Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. Conditional functional dependencies for data cleaning. In *Proceedings of the 23rd International Conference on Data Engineering*, ICDE '07, pages 746–755, 2007.

[15] Toon Calders and Jef Wijsen. On monotone data mining languages. In *Proceedings of the 8th International Workshop on Database Programming Languages*, DBPL '01, pages 119–132, 2001.

[16] Nathalie Caspard and Bernard Monjardet. The lattices of closure systems, closure operators, and implicational systems on a finite set: A survey. *Discrete Applied Mathematics*, 127(2):241–269, 2003.

[17] Brice Chardin, Emmanuel Coquery, Benjamin Gouriou, Marie Pailloux, and Jean-Marc Petit. Query Rewriting for Rule Mining in Databases. In Bruno Crémilleux, Luc De Raedt, Paolo Frasconi, and Tias Guns, editors, *Languages for Data Mining and Machine Learning (LML) Workshop@ECML/PKDD 2013*, pages 1–16, September 2013.

[18] Brice Chardin, Emmanuel Coquery, Marie Pailloux, and Jean-Marc Petit. RQL: An SQL-like Query Language for Discovering Meaningful Rules (demo). In *IEEE ICDM 2014, Shengzen, China*, December 2014.

[19] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. Discovering denial constraints. *Proc. VLDB Endow.*, 6(13):1498–1509, August 2013.

[20] János Demetrovics and Vu Duc Thi. Some remarks on generating Armstrong and inferring functional dependencies relation. *Acta Cybernetica*, 12(2):167–180, 1995.

[21] Wenfei Fan and Floris Geerts. Uniform dependency language for improving data quality. *IEEE Data Eng. Bull.*, 34(3):34–42, 2011.

[22] Lujun Fang and Kristen LeFevre. Splash: ad-hoc querying of data and statistical models. In *Proceedings of the 13th International Conference on Extending Database Technology*, EDBT '10, pages 275–286, 2010.

[23] Peter A. Flach and Iztok Savnik. Database dependency discovery: A machine learning approach. *AI Commun.*, 12(3):139–160, 1999.

[24] Frédéric Flouvat, Fabien De Marchi, and Jean-Marc Petit. The izi project: Easy prototyping of interesting pattern mining algorithms. In *New Frontiers in Applied Data Mining*, volume 5669 of *LNCS*, pages 1–15. Springer, 2010.

[25] Michael L. Fredman and Leonid Khachiyan. On the complexity of dualization of monotone disjunctive normal forms. *J. Algorithms*, 21(3):618–628, 1996.

[26] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis*. Springer, 1999.

[27] Lukasz Golab, Howard J. Karloff, Flip Korn, Avishek Saha, and Divesh Srivastava. Sequential dependencies. *PVLDB*, 2(1):574–585, 2009.

[28] Lukasz Golab, Howard J. Karloff, Flip Korn, and Divesh Srivastava. Data auditor: Exploring data quality and semantics using pattern tableaux. *PVLDB*, 3(2):1641–1644, 2010.

[29] Georg Gottlob and Leonid Libkin. Investigations on Armstrong relations, dependency inference, and excluded functional dependencies. *Acta Cybernetica*, 9(4):385–402, 1990.

[30] Jean-Louis Guigues and Vincent Duquenne. Familles minimales d'implications informatives résultant d'un tableau de données binaires. *Math. Sci. Humaines*, 24(95):5–18, 1986.

[31] Tias Guns, Siegfried Nijssen, and Luc De Raedt. Itemset mining: A constraint programming perspective. *Artif. Intell.*, 175(12-13):1951–1983, 2011.

[32] Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. TANE: an efficient algorithm for discovering functional and approximate dependencies. *Comput. J.*, 42(2):100–111, 1999.

[33] Tomasz Imielinski and Heikki Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39(11):58–64, 1996.

[34] Stephen S. Intille, Kent Larson, Emmanuel Munguia Tapia, Jennifer S. Beaudin, Pallavi Kaushik, Jason Nawyn, and Randy Rockinson. Using a live-in laboratory for ubiquitous computing research. In *PERVASIVE '06*, pages 349–365, 2006.

[35] Nick Koudas, Avishek Saha, Divesh Srivastava, and Suresh Venkatasubramanian. Metric functional dependencies. In *ICDE*, pages 1275–1278, 2009.

[36] Moshe Lichman. UCI machine learning repository, 2016.

[37] Stéphane Lopes, Jean-Marc Petit, and Lotfi Lakhal. Efficient discovery of functional dependencies and Armstrong relations. In *EDBT 2000*, pages 350–364, 2000.

[38] Stéphane Lopes, Jean-Marc Petit, and Lotfi Lakhal. Functional and approximate dependency mining: database and FCA points of view. *J. Exp. Theor. Artif. Intell.*, 14(2-3):93–114, 2002.

[39] Heikki Mannila and Kari-Jouko Räihä. Algorithms for inferring functional dependencies from relations. *Data and Knowldege Engineering*, 12(1):83–99, 1994.

[40] Heikki Mannila and Hannu Toivonen. Levelwise search and borders of theories in knowledge discovery. *DMKD*, 1(3):241–258, 1997.

[41] MarketResearch.com. Worldwide relational database management systems 2012-2016 forecast, Aug 2012.

[42] Raoul Medina and Lhouari Nourine. A unified hierarchy for functional dependencies, conditional functional dependencies and association rules. In *ICFCA*, volume 5548 of *LNCS*, pages 98–113. Springer, 2009.

[43] Rosa Meo, Giuseppe Psaila, and Stefano Ceri. An extension to SQL for mining association rules. *Data Mining and Knowledge Discovery*, 2(2):195–224, 1998.

[44] Jean-Philippe Métivier, Patrice Boizumault, Bruno Crémilleux, Mehdi Khiari, and Samir Loudni. A constraint-based language for declarative pattern discovery. In *Declarative Pattern Mining Workshop, ICDM 2011*, pages 1112–1119, 2011.

[45] Keisuke Murakami and Takeaki Uno. Efficient algorithms for dualizing large-scale hypergraphs. *CoRR*, 1102.3813, 2011.

[46] Benjamin Négrevergne, Alexandre Termier, Marie-Christine Rousset, and Jean-François Méhaut. Para miner: a generic pattern mining algorithm for multi-core architectures. *Data Min. Knowl. Discov.*, 28(3):593–633, 2014.

[47] Lhouari Nourine and Jean-Marc Petit. Extending Set-Based Dualization: Application to Pattern Mining. In IOS Press, editor, *ECAI 2012*, August 2012.

[48] Lhouari Nourine and Jean-Marc Petit. Extended dualization: Application to maximal pattern mining, Theor. Comput. Sci., 618, pages 107–121, 2016

[49] Carlos Ordonez and Sasi K. Pitchaimalai. One-pass data mining algorithms in a DBMS with UDFs. In *SIGMOD Conference*, pages 1217–1220, 2011.

[50] Thorsten Papenbrock, Tanja Bergmann, Moritz Finke, Jakob Zwiener, and Felix Naumann. Data profiling with metanome. *PVLDB*, 8(12):1860–1871, 2015.

[51] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. Functional dependency discovery: An experimental evaluation of seven algorithms. *PVLDB*, 8(10):1082–1093, 2015.

[52] Jonathan Vincent, Pierre Martre, Benjamin Gouriou, Catherine Ravel, Zhanwu Dai, Jean-Marc Petit, and Marie Pailloux. RulNet: A Web-Oriented Platform for Regulatory Network Inference, Application to Wheat –Omics Data. *PlosOne*, 10(5):20, May 2015.

[53] Jef Wijsen. Temporal dependencies. In *Encyclopedia of Database Systems*, pages 2960–2966. 2009.