RQL: un langage “à la SQL” pour découvrir des règles à partir des données

Brice Chardin
LIAS, ISAE-ENSMA
France

Emmanuel Coquery
Université Lyon 1
France

Marie Pailloux
Université Blaise Pascal
France

Jean-Marc Petit
INSA Lyon
France

ABSTRACT

RQL (pour *Rule Query Language*) est un langage de requêtes “à la SQL” qui étend et généralise les dépendances fonctionnelles à de nouvelles catégories de règles. RQL apporte aux analystes de données un outil pratique pour découvrir les implications logiques entre attributs d'une base de données. Ces implications peuvent mettre en évidence des problèmes de qualité de données ou de nouvelles corrélations inattendues entre les attributs. Le traitement de ces requêtes RQL est basé sur une technique de réécriture qui délègue un maximum de calculs au SGBD sous-jacent. Cette contribution vise à renforcer le lien entre la fouille de données et les bases de données et de faciliter l'utilisation de techniques de fouille par des analystes ou des étudiants habitués au SQL.

1. INTRODUCTION

Pattern mining can be seen as an automated part of data exploration. For instance, functional dependencies or conditional functional dependencies are definitely useful to understand the data and to identify data quality problems [5]. However, pattern mining techniques are rarely usable directly by data analysts. Most of the time, they have to perform some data pre-processing between different systems and formats. The pattern mining codes themselves often require to be compiled from some specific programming languages. All these steps are out of reach of many data analysts, rending round-trip engineering into a nightmare. Automated rule generation can also flood the analyst with huge amounts of patterns, and make it difficult to extract useful information. Other techniques have to be provided to interact with the data and give useful feedback to the analysts.

Demo contribution To improve pattern mining usability for data exploration, we introduce a Rule Query Language (RQL) that allows SQL-aware analysts to use pattern mining techniques with an interactive, user-friendly interface. In this demonstration, we show the usability of this web interface with the point of view of a data analyst. We focus on the expressive power of RQL through various examples, showing how easy it is to devise new and surprising rules with a very simple language derived from SQL. We also introduce how the data analysts can interact with the system through RQL queries and counterexamples taken from the database. During the demo, participants will be invited to formulate their own queries on predefined databases to discover attribute relationships through generated rules and counterexamples.

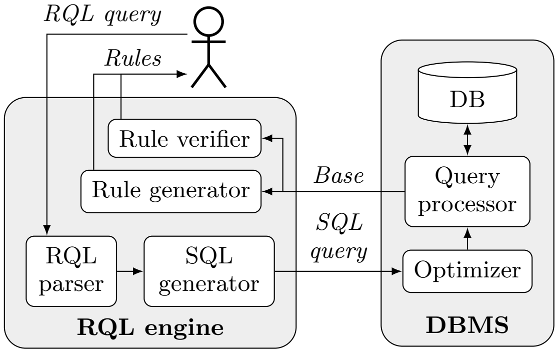


Figure 2: RQL queries processing overview

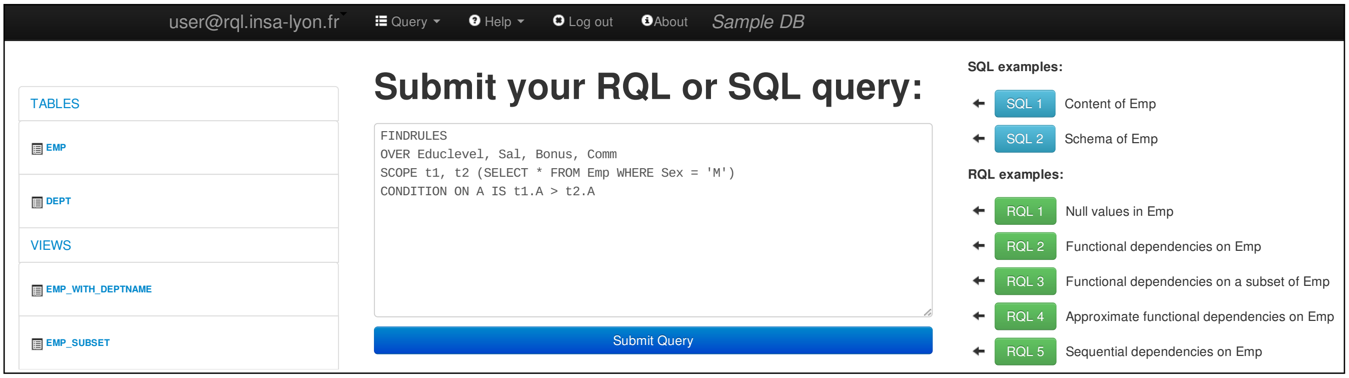
Figure 1 gives a preview of the web interface for RQL, made available[1] for research and educational purposes. This interface provides an unified access to the user's data and pattern mining techniques using declarative languages: SQL and RQL.

From previous works [1, 2, 6], RQL is compliant with Armstrong's axioms, i.e. the language generalizes functional dependencies to a new class of dependencies based on logical implications (if ... then statements). To the best of our knowledge, this class of dependencies has not been studied before. We have proven that these dependencies can be efficiently computed from a database using a two step technique. First, a non trivial SQL subquery is generated to compute a *base* of the associated closure system – a base is also called a context in formal concept analysis terminology [8]. Then, a state of the art algorithm [12] is used to generate a canonical cover of rules from this base. This approach allows RQL to benefit from DBMSs' query optimization to access the data, and keep the data where they are.

Figure 2 gives an overview of this architecture with respect to RQL query processing. As for SQL queries, the application simply forwards them to the underlying DBMS, which makes the transition between SQL and RQL transparent to the user. The ultimate goal of this work is to integrate pattern mining techniques into core DBMS technologies [13].

Related works Defining specific languages for pattern mining is a long standing goal [3], for example using constraint programming techniques [10]. Nevertheless, we argue that

[1] `http://rql.insa-lyon.fr`


user@rql.insa-lyon.fr
Query
Help
Log out
About
Sample DB
TABLES
EMP
DEPT
VIEWS
EMP_WITH_DEPTNAME
EMP_SUBSET
Submit your RQL or SQL query:
Submit Query
SQL examples:
SQL 1 Content of Emp
SQL 2 Schema of Emp
RQL examples:
RQL 1 Null values in Emp
RQL 2 Functional dependencies on Emp
RQL 3 Functional dependencies on a subset of Emp
RQL 4 Approximate functional dependencies on Emp
RQL 5 Sequential dependencies on Emp

```
FINDRULES
OVER Educlevel, Sal, Bonus, Comm
SCOPE t1, t2 (SELECT * FROM Emp WHERE Sex = 'M')
CONDITION ON A IS t1.A > t2.A
```

Figure 1: Web interface for RQL

pattern mining languages should benefit from direct extensions of the SQL language, since data are often stored in DBMSs. Other practical approaches, as close as possible of DBMSs, have been proposed to interact more directly with DBMSs query engines [7, 14, 4].

2. THE RQL QUERY LANGUAGE

To make things concrete, let us consider the running example given in Figure 3 with the *EMP* table. The attribute *Educlevel* represents the number of years of formal education, *Sal* the yearly salary, *Bonus* the yearly bonus and *Comm* the yearly commission. The meaning of other attributes is straightforward.

To begin with, let us extract functional dependencies (FD) from the relation *Emp*. Recall that a FD $X \to Y$ holds in r if for all tuples $t1, t2 \in r$, and for all attribute $A \in X$ such that $t1[A] = t2[A]$ then for all $A \in Y$, $t1[A] = t2[A]$. With RQL, FDs are expressed in a similar way.

Example 1. Q_1 discovers FDs from *Emp* over a subset of attributes.

Q_1:
```
FINDRULES
OVER Empno, Lastname, Workdept, Job,
     Sex, Bonus, Mgrno
SCOPE t1, t2 Emp
CONDITION ON $A IS t1.$A = t2.$A
```

Note how the `CONDITION` clause matches the previous logical implication. We have also restricted FDs discovery to a subset of seven attributes in the `OVER` clause. In this example, a canonical cover of FDs that hold in *Emp* is generated (composed of twenty-four FDs), including FDs such as *Empno* → *Lastname* or *Workdept* → *Job*.

Overall, a RQL query has the following general form:

```
FINDRULES
OVER [set of attributes: $A_1$, ..., $A_n$]
SCOPE [tuple variables: $t_1$, ..., $t_n$]
WHERE [condition on ($t_1$, ..., $t_n$)]
CONDITION ON [attribute variable: $\$A$]
  IS [condition on ($\$A$, $t_1$, ..., $t_n$)]
```

The `FINDRULES` keyword identifies a RQL query, which generates rules of the form $X \to Y$ with X and Y disjoint attribute sets taken from the `OVER` clause. The `SCOPE` clause defines tuple-variables over some tables obtained by classical SQL queries. An optional `WHERE` clause defines relationships between tuple-variables, similar to the SQL `WHERE` clause. The `CONDITION ON $A` clause defines the predicate to be satisfied by each attribute `$A` occurring in the left- and right-hand sides of the rule.

To illustrate the expressiveness of RQL queries, we now provide several examples.

Example 2. Let us consider null values known to be common in real-life databases. With RQL, the data analyst has the opportunity to discover rules between attributes with respect to null values as shown with query Q_2.

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

The rule *Mgrno* → *Workdept* holds in *Emp* since each time the attribute *Mgrno* is null in a tuple, then *Workdept* is also null for the same tuple (only employee No. 20 in this example).

Note that the difference between Q_1 and Q_2 naturally lies on the predicate to be evaluated, but also on the number of tuple variables required. The predicate of Q_1 is evaluated on pairs of tuples, while Q_2 considers tuples individually.

Example 3. The following query Q_1' restricts the scope of Q_1, leading to the notion of conditional functional dependencies [5]. For example, we consider only employees with a level of qualification above 16.

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
```

Interestingly, *Sex* → *Bonus* holds with this restriction, meaning that above a certain level of qualification (16), the gender determines the bonus.

Example 4. Query Q_1'' is an approximation of Q_1 for numeric values similar to Metric Functional Dependencies [11], where strict equality is discarded to take into account variations under 10%. For instance, salaries 41250 and 38250

EMP	Empno	Lastname	Workdept	Job	Educlevel	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	700	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 3: Running example

are considered close (7.5% difference), but not salaries 41250 and 36170 (13.1% difference).

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
```

In that case, *Sal* → *Comm* holds, meaning that employees earning similar salaries receive similar commissions.

We have shown so far query examples related to implications (in FCA) and functional dependencies (in DB). Nevertheless, RQL is not restricted to these types of queries at all and can express many more rules.

Example 5. Assume we are interested in a kind of sequential dependencies [9], 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
```

Sal → *Comm* and *Comm* → *Sal* hold in *Emp*, which means that a higher salary is equivalent to a higher commission.

Example 6. Continuing the previous example, assume now the analyst wants to focus on male employees (see also Figure 1).

Q_3':
```
FINDRULES
OVER Educlevel, Sal, Bonus, Comm
SCOPE t1, t2 (SELECT * FROM Emp
      WHERE Sex='M')
CONDITION ON $A IS t1.$A > t2.$A
```

In that case, *Educlevel* → *Bonus* also holds, which means that male employees with higher education levels receive higher bonuses.

Example 7. Instead of narrowing the scope of a query, user-defined conditions can bind different tuple variables together with a custom relationship specified in the `WHERE` clause of the RQL query. For example, Q_4 finds disparities between managers and managees, i.e. rules on attributes for which managers have values greater than or equal to their managees.

Q_4:
```
FINDRULES
OVER Educlevel, Sal, Bonus, Comm
SCOPE t1, t2 Emp
WHERE t1.Empno = t2.Mgrno
CONDITION ON $A IS t1.$A >= t2.$A
```

In this example, $\emptyset \rightarrow$ *Bonus* holds in *Emp*, which means that managers always earn a bonus greater than or equal to their managees'.

3. FEEDBACK THROUGH COUNTEREXAMPLES

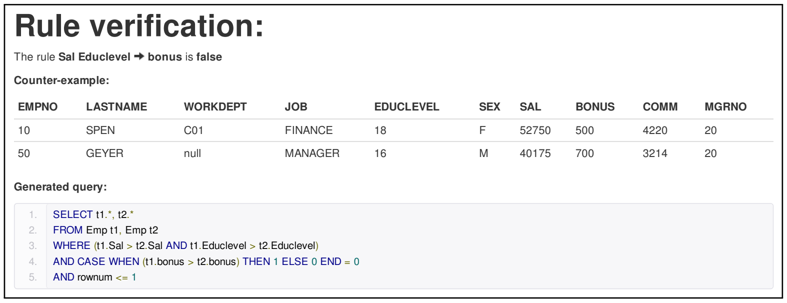
Given a RQL query, the data analyst may also interact with the system to know whether or not a given rule holds. She can provide a rule to the system and two cases arise: either the rule holds and the analyst is notified that the rule is indeed valid; or the rule does not hold which means that at least one counterexample exists and one of them is provided by the system. This notion of counterexample is well known for functional dependencies, and provides very good feedback to the data analyst with her own data.We strongly believe that counterexamples are a great tool to help the analyst understand why a particular rule does not hold, and refine if necessary her analysis in an iterative process.

To illustrate counterexamples, suppose that a data analyst wants to explore her hypothesis that higher salaries and higher education levels yield higher bonuses (*Salary*, *Educlevel* → *Bonus*), using Q_3. Figure 4 gives an overview of what RQL provides as a counterexample for this rule, that is, two tuples among which one (employee No. 10) has a higher *Salary* and *Educlevel* than the other (employee No. 50), but not a higher *Bonus*.

With this counterexample as a starting point, and especially the SQL query generated to extract it from the database, the data analyst can quickly switch to SQL to get an idea of why this rule is not verified. For instance, employees that are either female or have a finance job are easily pointed out as having a higher salary and education level, but lower bonuses than others. The data analyst can then refine her RQL query, for example by narrowing the scope of the data, such as in Q_3' where a higher *Educlevel* by itself implies a higher *Bonus*.

The number of tuples required to provide a counterexample depends on the number of tuple variables. Q_3 or FDs need at least two tuples. But with Q_2, one tuple (for instance employee No. 30) is enough to prove that the rule *Workdept* → *Mgrno* does not hold.

4. IMPLEMENTATION AND APPLICATION



Rule verification:

The rule **Sal Educlevel ➔ bonus** is **false**

Counter-example:

EMPNO	LASTNAME	WORKDEPT	JOB	EDUCLEVEL	SEX	SAL	BONUS	COMM	MGRNO
10	SPEN	C01	FINANCE	18	F	52750	500	4220	20
50	GEYER	null	MANAGER	16	M	40175	700	3214	20

Generated query:

```
SELECT t1.*, t2.*
FROM Emp t1, Emp t2
WHERE (t1.Sal > t2.Sal AND t1.Educlevel > t2.Educlevel)
AND CASE WHEN (t1.bonus > t2.bonus) THEN 1 ELSE 0 END = 0
AND rownum <= 1
```

Figure 4: Counterexample with RQL

The RQL web application has been implemented in Java with the Play Framework [15]. External tools have been used for the most expensive part of the rule generation process, i.e. the enumeration of minimal transversal of hypergraphs [12]. The chosen DBMS is Oracle 11g Release 2.

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 issue her own query to identify more counterexamples.

RQL can be interacted with in two modes: (i) a Sample DB is provided with selected examples to offer a quick way into RQL (ii) a Sandbox allows users to upload and query their own data (currently limited to 3 tables and 200 kB). RQL has been used by 120 undergraduate students to play with functional dependencies and other constraints in a database course at INSA Lyon. Not surprisingly, the notion of counterexamples has been widely used by students. RQL has been appreciated for its ability to bridge the gap between the SQL language and functional dependencies in database design by providing a unified interface for both SQL and RQL queries.

Previous works [6] have highlighted the efficiency of RQL as a two step process, even on large databases.

5. CONCLUSION

RQL is introduced as a web interface to discover rule patterns over relational databases. RQL subsumes SQL statements by providing the opportunity to specify and get results as a set of rules or some counterexamples. 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 [6]. RQL allows SQL developers to extract precise information without any specific knowledge in data mining.

6. REFERENCES

[1] M. Agier, C. Froidevaux, J.-M. Petit, Y. Renaud, and J. Wijsen. On Armstrong-compliant logical query languages. In *Proceedings of the 4th International Workshop on Logic in Databases*, LID '11, pages 33–40, 2011.

[2] M. Agier, J.-M. Petit, and E. Suzuki. Unifying framework for rule semantics: Application to gene expression data. *Fundamenta Informaticae*, 78(4):543–559, 2007.

[3] H. Blockeel, T. Calders, E. Fromont, B. Goethals, A. Prado, , and C. Robardet. A practical comparative study of data mining query languages. In Springer, editor, *Inductive Databases and Constraint-Based Data Mining*, pages 59–77. 2010.

[4] H. Blockeel, T. Calders, É. Fromont, B. Goethals, A. Prado, and C. Robardet. An inductive database system based on virtual mining views. *Data Min. Knowl. Discov.*, 24(1):247–287, 2012.

[5] P. Bohannon, W. Fan, F. Geerts, X. Jia, and A. Kementsietsidis. Conditional functional dependencies for data cleaning. In *Proceedings of the 23rd International Conference on Data Engineering*, ICDE '07, pages 746–755, 2007.

[6] B. Chardin, E. Coquery, B. Gouriou, M. Pailloux, and J.-M. Petit. Query Rewriting for Rule Mining in Databases. In *Languages for Data Mining and Machine Learning, in conjunction with ECML/PKDD*, pages 35–49, 2013.

[7] L. Fang and K. 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.

[8] B. Ganter and R. Wille. *Formal Concept Analysis.* Springer, 1999.

[9] L. Golab, H. J. Karloff, F. Korn, A. Saha, and D. Srivastava. Sequential dependencies. *PVLDB*, 2(1):574–585, 2009.

[10] T. Guns, S. Nijssen, and L. D. Raedt. Itemset mining: A constraint programming perspective. *Artif. Intell.*, 175(12-13):1951–1983, 2011.

[11] N. Koudas, A. Saha, D. Srivastava, and S. Venkatasubramanian. Metric functional dependencies. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, ICDE '09, pages 1275–1278, Washington, DC, USA, 2009. IEEE Computer Society.

[12] K. Murakami and T. Uno. Efficient algorithms for dualizing large-scale hypergraphs. *CoRR*, 1102.3813, 2011.

[13] A. Netz, S. Chaudhuri, J. Bernhardt, and U. M. Fayyad. Integration of data mining with database technology. In *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB '00, pages 719–722, 2000.

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

[15] Play framework. http://www.playframework.com/.