Towards Evolutionary, Domain-Specific Query Classification Based on Policy Rules

Peter K. Schwab and Klaus Meyer-Wegener

Friedrich-Alexander-Universität Erlangen-Nürnberg
{firstname.lastname}@fau.de, https://www.cs6.tf.fau.eu/

Abstract. Many devices like smart sensors produce a vast amount of data that are still commonly stored in relational databases and are being processed using SQL queries. This data is only useable if it is processed in a fashion that results in applicable information for the users posing these queries. Thus, it can be very supportive for them to assess other queries that have already processed the targeted data. This is not a simple exercise, as SQL allows alias names and various syntactic structures to express equivalent queries. A manual assessment is also hard to accomplish due to the amount of qualified queries. We present a framework for evolutionary SQL query classification. Based on the analysis of query logs, query metadata like schema lineage and result statistics are automatically derived. Our framework enables users to define domain-specific policy rules for automatic query classification based on the query metadata. Classification is done according to domain-specific, contextual attributes that can be defined evolutionary at runtime, together with the policy rules. The classification results enrich the query metadata.

1 Introduction

Hoarding vast amounts of data is no longer a big thing to undertake, for example by smart sensors in the context of Industry 4.0. Instead, the key task is to extract the desired information from the data for a particular purpose at a certain point in time [3]. Data is only useable when accessed in a fashion that results in applicable information for the users posing the accessing queries. Thus, it can be very supportive for them to assess other queries that already have accessed the targeted data set. Most data are still commonly stored in relational databases (DBs) and are being processed using SQL queries. Analyzing these queries regarding their kind of data access is not a simple exercise, as SQL allows alias names and various syntactic structures for equivalent queries (e. g. subquery instead of join). Query assessment can be supported by considering query metadata (QM). A manual assessment is often hard to accomplish because of the vast amount of qualified queries. In addition, most query-assessment results are not commonly accessible but only available as tacit knowledge in the heads of the resp. users.

Problem Statement. We require novel approaches that support query assessment according to a certain processing context. They must ease the analysis of the SQL queries' syntactical variety and support automation of the query classification. Users must be enabled to store their assessment results linked with the underlying queries in order to share them with other users.

Contribution. We present a framework for evolutionary, domain-specific SQL query classification based on policy rules. QM like schema lineage and result statistics are automatically derived based on the analysis of query logs. Our framework enables users to externalize their tacit knowledge into domain-specific policy rules for automatic query classification based on the QM. Classification results are stored in contextual QM attributes (QMAs). They can be used for further classification in other policy rules. The contextual QMAs as well as the policy rules can be defined evolutionary at runtime.

2 Policy-Based Query Classification

We provide a policy-based, automatic query classification [8] based on relational and graph-based data models holding QM [7].

Evolutionary Definition of Contextual Query Metadata. Examples for contextual QM are a query's purpose, its compliance according to data-privacy directives, or its aptitude for hardware acceleration. So far, this type of QM was mapped to our relational model. We extend our multi-relational property-graph model [7] to enable the evolutionary definition of contextual QM at runtime. For every query, a graph with a root vertex holding a UID is modeled. This root can have several edges of type hasContextualAttr to vertices of type contextualAttibute. A vertex has two properties holding the name and the value of the resp. contextual QMA. In addition, a contextualAttibute vertex has exactly one edge hasDataType to a vertex of type dataType. To support a slender, user-centric set of data types that can be selected for contextual QMA, we orientate towards the requirements interchange format (ReqIF) as a standard for user-centric types that fulfill an end-user's plain idea of data types [2] and provide *Boolean*, *String*, *Integer*, *Float*, *Timestamp*, and *Enumeration*, and lists of these data types.

Domain-Specific Policy Rules. Our policies are based on conditional rules. The Boolean expression in their antecedent part describes a query-processing pattern. We provide a domain-specific language to write it down [8]. Our query representation is independent from SQL syntax using the queries' corresponding trees of relational algebra operators. A single tree covers many syntactical variants of semantically equivalent queries. This syntax-independent representation enables a more generic definition of patterns. A *basic* pattern $\mathsf{p}_{\mathsf{basic}}$ is related to a single QM attribute and covers for example an accessed relation or schema attribute, a certain filter predicate, the related DB user, or a query's runtime or number of result tuples. Schema lineage is resolved automatically. Up to now, basic patterns have been combined by logical conjunction to a *complex* pattern p_{complex}. We extend the combination possibilities by adding Boolean operators (NOT, OR, XOR) and nesting via parentheses to create richer complex patterns.

Queries matching a complex pattern will be classified according to the rule's consequent part. So far, this part was fixed on the contextual QMA data-privacy compliance. Based on our data-privacy use case [6], any policy rule classified a matching query q always as *non-compliant* (cf. List. 1, line 2).

We extend our policy rules' consequent part and allow classification to arbitrary contextual QMAs. Now both contextual QMAs and policy rules can be defined at runtime. The assigned value v has to match the data domain of the related contextual QMA qma_c (cf. List. 1, line 5). When q is classified, its contextual QM is enriched with the classification result and the related policy IDs of all matching rules. This enables traceability of the classification process.

```
1  /* Status Quo of Policy-Rule Definition */
2  IF q.match( p_complex ) THEN q.classify( ‘non-compliant’ )
3
4  /* Evolutionary Policy Rules */
5  IF q.match( p_complex ) THEN q.classify( qma_c, v )
```

Listing 1. Status quo of our policy rules and the proposed extension.

3 Exemplary Classification Use Case

Up to now, our approach was tailored towards the use case of data-privacy compliance [6,7,8]. Examples for policy rules in this context are the prohibition of filters on certain personal data or the requirement of a minimum result size in order to prevent users from drawing conclusions on individuals by queries. We will motivate now another use case that is totally different to the present one in order to demonstrate that by enabling evolutionary contextual QMAs, policy-based query classification can be applied in arbitrary scenarios without the need of adapting our implementation.

To enable hardware-based acceleration of DB query processing, the project "Reconfigurable Data Provider (ReProVide)" provides a sophisticated storage solution based on field-programmable gate arrays (FPGAs) [1]. Its query optimization techniques consider the capabilities of the hardware for a scalable and highly performant near-data processing of Big Data [5]. ReProVide's generic FPGA architecture offers a library of query-processing modules, which can be configured onto the FPGAs. So far, queries apt for near-data processing are selected manually based on tacit expert knowledge. The ReProVide system is a system on a chip (SoC) with its own storage [4]. Only queries accessing data that are stored there can be accelerated by the FPGAs. Filter operators, for example, can be accelerated at line rate. But there are different query-processing modules for filters, depending on the involved data type. Thus, assuming that the date dimension of the TPC-DS benchmark suite[1] is located on the RePro-

[1] http://www.tpc.org/tpcds/

Vide storage, a responsible DB administrator could first create a new contextual QMA at runtime with name hardware-acceleration aptitude and data type *List<Enumeration>* with the elements {'filter (float)', 'filter (int)', 'filter (uint)', 'filter (boolean)', 'filter (string)', 'filter (date)', 'filter (timestamp)'}. Then, the admin could create the policy rule shown in List. 2 which triggers automatic classification of queries containing filter operations on integers. The admin accordingly creates further policy rules covering filter operations on other data types. This means, a query containing several filter operations on different data types can be classified by different policy rules. Therefore, our contextual QMA was defined as a List. For example, the query in List. 3 will finally be classified as hardware-acceleration aptitude = {'filter (int)', 'filter (string)'}.

```
IF q.match(
  restrictsOn('date_dim', 'd_year') OR
  restrictsOn('date_dim', 'd_dow') OR
   ...
  restrictsOn('date_dim', 'd_last_dom')
)
THEN q.classify('hardware-acceleration aptitude',
                 'filter (int)'
)
```

Listing 2. Example rule to classify queries apt for hardware acceleration.

```
SELECT d_year, d_dow
FROM   date_dim
WHERE  d_day_name="Monday" AND (d_year > 1900 OR d_moy > 4)
```

Listing 3. Example query that filters on data stored within the ReProVide system.

As ReProVide also allows hardware acceleration of projections and semi-joins, the enumeration's elements of our contextual QMA can be extended by respective elements and new policy rules could be defined to enable classification of queries containing these operations. All of this can happen at runtime, without adapting our implementation. Furthermore, the authors of ReProVide also aim query-sequence optimization [4]. Our framework can also support this aim by defining additional contextual QMAs and policy rules – again at runtime.

4 Next Steps

We will elaborate the use case for hardware acceleration in more detail concerning our approach. Furthermore, we have to solve the problem of contradicting classification results based on conflicting policy rules. A prototypic implementation will give further information about the applicability of our evolutionary approach for arbitrary scenarios.

Acknowledgement: The authors would like to thank the anonymous reviewers for their valuable remarks.

References

1. Becher, et al.: Reprovide: Towards utilizing heterogeneous partially reconfigurable architectures for near-memory data processing. In: BTW, 18. Fachtagung des GI-Fachbereichs DBIS, Workshopband. LNI, vol. P-290, pp. 51–70. GI, Bonn (2019)
2. Ebert, et al.: ReqIF: Seamless requirements interchange format between business partners. IEEE Softw. **29**(5) (2012)
3. Lee, et al.: Recent advances and trends in predictive manufacturing systems in big data environment. Manufacturing letters **1**(1) (2013)
4. Lekshmi B. G., et al.: The ReProVide query-sequence optimization in a hardware-accelerated DBMS. In: 16th Int. Workshop DaMoN. pp. 17:1–17:3. ACM (2020)
5. Lekshmi B. G., et al.: SQL query processing using an integrated FPGA-based near-data accelerator in ReProVide. In: Proc. 23nd Int. Conf. EDBT. pp. 639–642. OpenProceedings.org (2020)
6. Schwab, et al.: Query-driven enforcement of rule-based policies for data-privacy compliance. In: Proc. LWDA (2019)
7. Schwab, et al.: A framework for DSL-based query classification using relational and graph-based data models. In: Proc. Joint Wksh. GRADES-NDA. ACM (2020)
8. Schwab, et al.: We know what you did last session – policy-based query classification for data-privacy compliance with the DataEconomist. In: Proc. SSDBM (2020)