Query-Driven Data Minimization with the DataEconomist

Peter K. Schwab
Friedrich-Alexander-Universität Erlangen-Nürnberg
peter.schwab@fau.de

Julian O. Matschinske
Friedrich-Alexander-Universität Erlangen-Nürnberg
julian.matschinske@fau.de

Andreas M. Wahl
Friedrich-Alexander-Universität Erlangen-Nürnberg
andreas.wahl@fau.de

Klaus Meyer-Wegener
Friedrich-Alexander-Universität Erlangen-Nürnberg
klaus.meyer-wegener@fau.de

ABSTRACT

In this paper, we explain the demonstration of the DataEconomist, a framework for query-driven data minimization in relational DBMS conformable to law. Our approach automatically minimizes user access rights based on the analysis of SQL query logs and thereby enables a parsimonious data processing. A minimization of data collection is reached by automatic detection and manual deletion of data belonging to unneeded schema elements.

In contrast to existing approaches, the DataEconomist supports privacy officers by focusing on the queries instead of making them trawl through a vast amount of collected personal data and specify legal use cases for their processing. SQL queries easily show who has processed when which data for which purpose and make it easy for the privacy officers to decide about the queries' compliance with data privacy regulations. Our framework does not require any knowledge of SQL by providing a graphical tool for searching and filtering queries and visualizations of query result sets.

1 INTRODUCTION

The new EU General Data Protection Regulation (GDPR) has an impact on every organization around the world [16], but most of them are prepared inadequately and are not aware of upcoming legal requirements [4]. They face the challenge of adapting their handling of personal data to become compliant to the contemporary requirements of data privacy. Art 5(1) GDPR postulates data minimization, which is one of the six general data-protection principles of the GDPR [1]. It requires that all collection and processing of personal data is only for a specific purpose, and that the quantity of these data is kept down as much as possible, so they are stored only as long as it is required to reach the intended purpose [15]. There are approaches like privacy by design [17] that include privacy protection in the overall conception of technical systems, but the vast majority of current systems does not consider these approaches. In order to ensure conformance to the principle of data minimization it is not sufficient to analyze the collected data sets only. This does not provide a possibility to verify that the data processing is exclusively for a specific purpose. But the responsible privacy officers often lack technical knowledge, which is usually required for the evaluation of data processing.

Problem Statement. In order to enable privacy officers to ensure data privacy in established systems, novel approaches are required, which provide mechanisms that do not premise profound technical knowledge to determine who collects personal data from where and who processes when which data for which purpose. The mechanisms must provide user-friendly interfaces and a data-processing description close to natural language [7].

Contribution. We demonstrate an extensible framework for query-driven data minimization in existing IT landscapes. We analyze SQL query logs and enrich queries with meta-information about the query structure, their environment, their execution, and their context. In contrast to currently trending approaches, queries are our first-class citizens for analysis of conformance to data privacy, instead of trawling through data stores that harbor vast amounts of personal data. Our framework aims to complement such approaches and supports privacy officers in minimizing user access to personal data, minimizing the storage of personal data to what is really necessary, and preventing future collection of unnecessary data.

Our framework is minimally-intrusive because it has no impact on productive operations. It can serve as a foundation of a system that automatically identifies unneeded schema relations and attributes in target databases (DBs), based on the queries running on these systems. Queries can be classified manually as legal or illegal regarding data-privacy regulations. User access rights can automatically be customized, and unnecessary data can be automatically deleted from the target DBs. Furthermore, we list the queries and the related users that have inserted unneeded data into the target DBs.

This paper describes the client-server architecture of our framework (cf. sec. 2.1), illustrates the conceptual schema for the query characteristics, i. e. the queries and their meta-information (cf. sec. 2.2), and outlines the reference implementation of the DataEconomist (cf. sec. 2.3). In [12], we have already described a user story from a healthcare scenario to emphasize our approach's benefits for ensuring data privacy, especially data minimization.

2 SYSTEM OVERVIEW

The software system is implemented with a client-server architecture using Web technologies. An end user can have three different roles: administrator, DB user, and privacy officer. Each of them needs to interact with the system, so it has to be accessible from different personal machines. Splitting the application into a client and server part allows for that and additionally simplifies initial setup for end users.

2.1 Client-Server Architecture

Internally, the DataEconomist server uses a PostgreSQL DB to persist query characteristics. It also connects with target DBs when needed to retrieve meta schemas required to understand the queries and execute them when requested. It supports all major relational DBMS and the Java JDBC API as well as standard SQL and several of its dialects.

The multi-user Web client is platform independent and does not require any kind of installation for the end user. It communicates with the server over HTTP using a RESTful JSON API.

The backend is implemented in Java using Apache Calcite [6] and the Spring Framework[1]. Fig. 1 shows its architecture.

The *Query Manager* and the *Database Manager* take care of ensuring consistency between main-memory representations of Query and Database objects and their DB entries. We must store additional information about these objects to control the redundancy. For queries, this information includes syntactical correctness, the number of restrictions, compliance with user privileges, and other potentially interesting properties. For target DBs, we fetch and store the schema in Java objects on the server.

The *Query Controller* and the *Database Controller* perform actions requested by the API on the resp. objects. They can only cause updates on the internal DB by using a manager module.

The *Search Module* maps search requests from the visual search form onto the internal DB. On top of that, it makes use of the *Query Manager* to consider information only available on the server application.

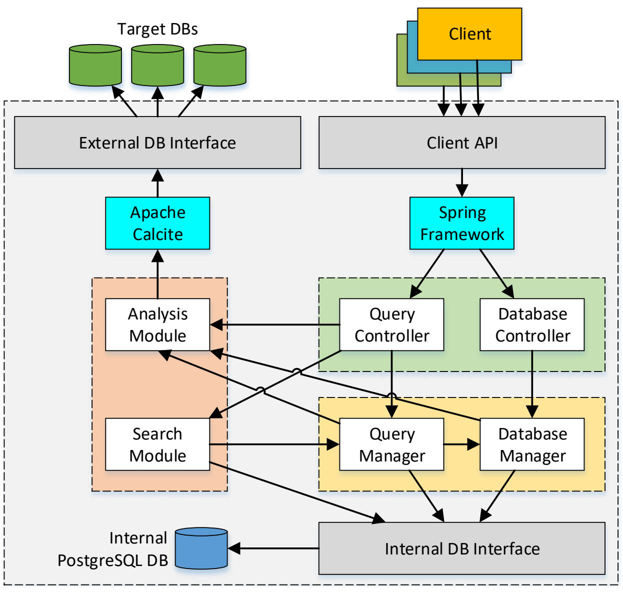


Figure 1: Architecture of the DataEconomist

2.2 Conceptual Schema

Fig. 2 shows an Entity-Relationship diagram containing the DataEconomist's conceptual schema to hold the queries and their meta-information. The Query entity is identified by the Text attribute, holding the query text. The complex, multi-valued attribute Relations stores the query's relations and attributes. It contains the two nested attributes RelName and Attributes. For each relation, we store its name in RelName and the names of all the relation's attributes in the multi-valued attribute Attributes.

The purpose of the Project entity is to group several target DBs in order to analyze their queries collectively. The Database entity holds connection information to target DBs.

A Query Execution reflects the run of a certain Query by a certain User on a certain Database at a certain Timestamp. The entity's further attributes are the RunTime in case of a successful query execution and the ErrorMessage if the query failed.

The Query Context points out a user's intention or the situation in which a Query Execution happens. Regarding this, privacy officers can specify the query's priority and its legitimacy concerning data privacy in this context. A query execution can only happen in one context, but a context may contain many query executions.

Fig. 2 highlights four groups of query characteristics in color. Some of them can be derived automatically; others must be entered manually. The Query Environment group is derived from the user and her initial connection to the target DB. The Query Execution group bundles when which query ran with what runtime successfully or not. Finally, the Query Context group cannot be derived automatically at all; it must be entered manually as it externalizes tacit user knowledge.

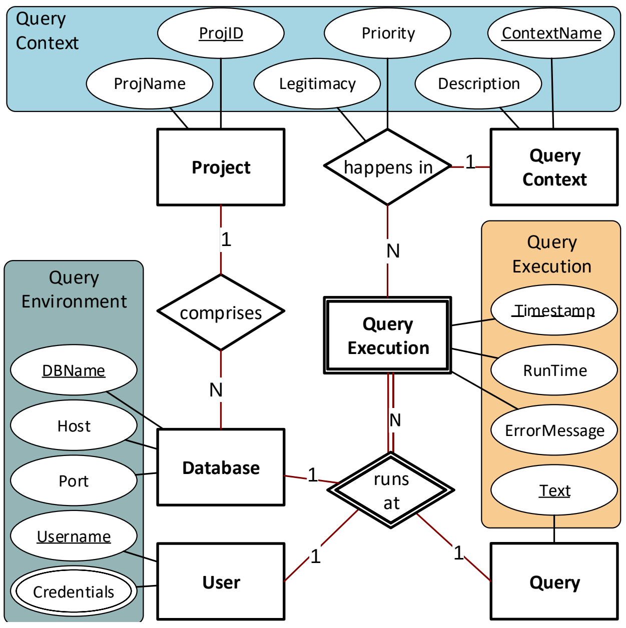


Figure 2: Conceptual schema of the DataEconomist

2.3 Reference Implementation

As first step, Apache Calcite performs basic syntax checking for each query. When checking which tables and columns are used by a query (a task that belongs to the core functionality of our DataEconomist), syntax alone is not sufficient. We additionally need the schema information. Asterisk selections are a trivial example to illustrate that. In order to detect used schema objects, Calcite's query planner transforms the query into a logical query plan. Calcite performs a query optimization and normalization. By applying the visitor pattern, the query-plan object is traversed and all used tables and columns are detected and collected for later review.

To provide the HTTP API, we use the model-view-controller module of the Spring framework. It maps HTTP requests onto the corresponding Java methods and serves as a bridge between Java objects and their JSON representation. The API mainly follows a CRUD approach, offering endpoints for entity creation, retrieval, update, and deletion.

[1]http://spring.io/projects/spring-framework

The frontend is implemented in Angular[2], a widely used Web framework written in TypeScript. It allows for the use of advanced GUI techniques such as reusable components and programming patterns such as reactive programming.

Fig. 3 shows the functionalities of the frontend interface. They are partitioned in three main views for projects, target DBs, and queries. The admin role can create projects and relate one or several target DBs. The admin needs to specify credentials for the target DBs so that the DataEconomist can connect to them and can fetch their schema information and import query logs. The admin role can also add and modify queries in our frontend, whereas the DB-user role can only choose a project and run the related queries.

Privacy officers can browse through all queries. We support them with a tool for extensive search-query formulation that can be utilized with purely visual means and does not require any knowledge of SQL. They can span a search-query tree by arbitrarily nesting search conditions concerning query structure or query characteristics. Any result set of queries can be manually filtered to be used in the next step for automatic detection of unneeded schema elements. Privacy officers can then check and adjust the DataEconomist's proposal of user-access-right minimization. The final configuration can be enforced to the target DBs with a single click.

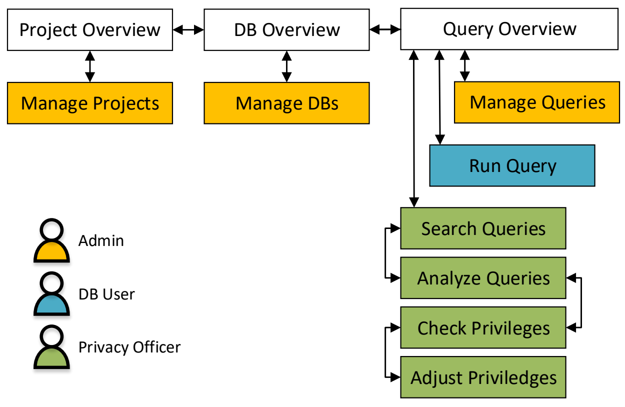


Figure 3: Frontend Use of the DataEconomist

3 DEMONSTRATION WALKTHROUGH

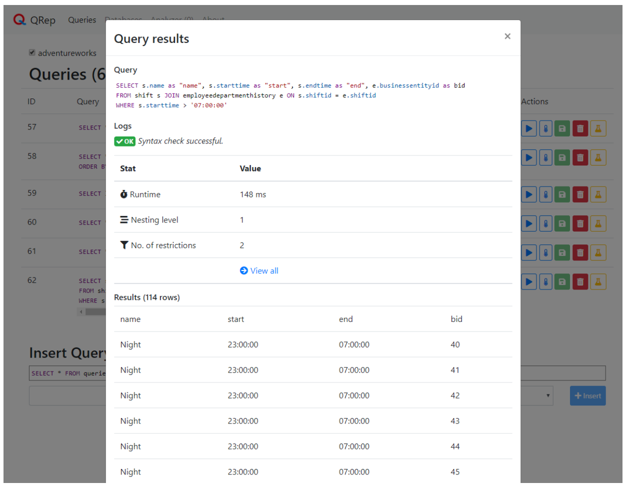
For this demonstration, we will shortly present our approach's fundamental concepts to the conference attendees. Then they will be briefed to use our framework and learn about various aspects of the current version of our DataEconomist. Attendees may go through three different demonstration scenarios that build upon each other, but do not require the preceding steps for understanding. Most steps in the process include interaction with the attendees.

Setting up a project and analyzing queries. We have prepared a substantial SQL query log containing queries of different complexities from an enterprise scenario, stored in a MS-SQL DB. The attendees can choose the queries to be investigated. When a query is added to the DataEconomist, it will automatically start to find various properties of that query and to make them available for a potential search (see next scenario).

[2] https://angular.io/

After the analyzing step is finished, attendees can see its results. The derived properties include the number of restrictions, the level of nesting, the number of different columns used by a query, and more.

Along with these properties, a sample of the query result is shown as well to help understand the query (see Fig. 4).



Query results

Query

```
SELECT s.name as "name", s.starttime as "start", s.endtime as "end", e.businessentityid as bid
FROM shift s JOIN employeedepartmenthistory e ON s.shiftid = e.shiftid
WHERE s.starttime > '07:00:00'
```

Logs

✓ OK *Syntax check successful.*

Stat	Value
Runtime	148 ms
Nesting level	1
No. of restrictions	2
	View all

Results (114 rows)

name	start	end	bid
Night	23:00:00	07:00:00	40
Night	23:00:00	07:00:00	41
Night	23:00:00	07:00:00	42
Night	23:00:00	07:00:00	43
Night	23:00:00	07:00:00	44
Night	23:00:00	07:00:00	45

Figure 4: Query stats and results dialog

Finding potentially illegal queries. For this scenario, we challenge the attendees to play the role of a privacy officer and to find queries violating data-privacy regulations. The attendees can use the visual-search interface and track down queries that retrieve for example data revealing racial or ethnic origin, political opinions, or religious or philosophical beliefs. To achieve a high level of flexibility, the visual-search queries can be nested and combined with Boolean operators (see fig. 5).

The attendees can directly consider accessible properties, such as the names of DB users, query-execution times, SQL strings, and all other query characteristics mentioned before.

We will examine two different cases that require different approaches. One requires the correct choice of properties to check. The other additionally requires a correct combination of properties. Both demonstrate the possibilities offered by the search-query formulation implemented in the DataEconomist.

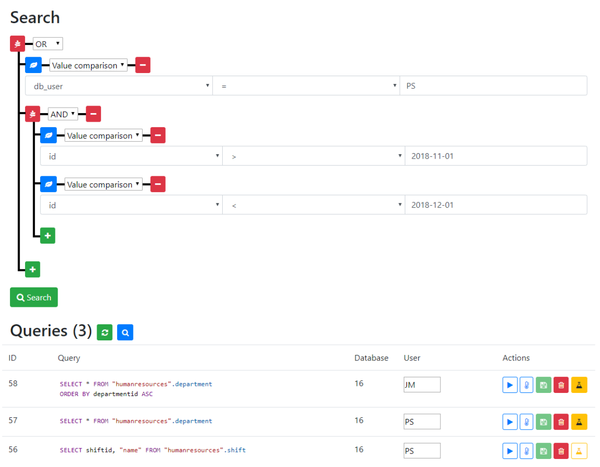
Minimizing user privileges and deleting unnecessary data. When illegal queries have been found, we demonstrate how the DataEconomist displays which tables and columns are used by a query or a set thereof (see Fig. 6).

Attendees can then adapt the user privileges by unchecking tick boxes at all schema objects the user should no longer have access to. These user privileges will actually be enacted on the MS-SQL server through the interface. To show the effects, we will then try to execute an illegal query on the DB, which will fail, and inform the user about the illegal access attempt.

After that, attendees can use the DataEconomist to automatically determine unnecessary schema elements, i. e. data that have no user access to them. Again, by a single click, all these data will be deleted on the MS-SQL server.

4 STATE OF THE ART

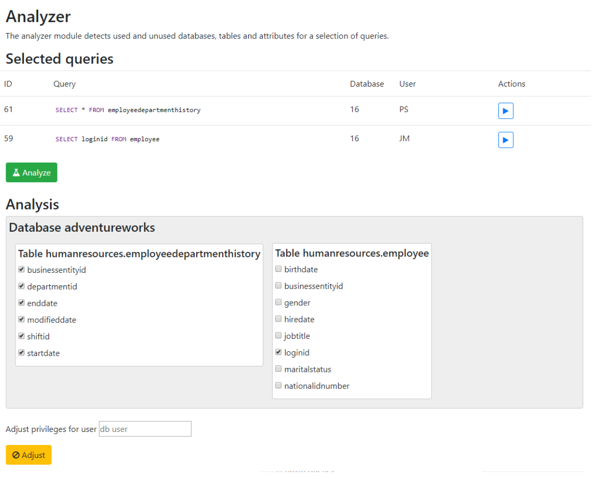
Srivastava et al. provide a relational framework for managing queries [5]. We adopt their way of administrating queries together with their meta-information in a relational model. There are



OR
Value comparison: db_user = PS
AND
Value comparison: id > 2018-11-01
Value comparison: id < 2018-12-01

ID	Query	Database	User	Actions
58	`SELECT * FROM "humanresources".department ORDER BY departmentid ASC`	16	JM	
57	`SELECT * FROM "humanresources".department`	16	PS	
56	`SELECT shiftid, "name" FROM "humanresources".shift`	16	PS	

Figure 5: Visual query search form



The analyzer module detects used and unused databases, tables and attributes for a selection of queries.

ID	Query	Database	User	Actions
61	`SELECT * FROM employeedepartmenthistory`	16	PS	
59	`SELECT loginid FROM employee`	16	JM	

Database adventureworks

Table humanresources.employeedepartmenthistory
- ✓ businessentityid
- ✓ departmentid
- ✓ enddate
- ✓ modifieddate
- ✓ shiftid
- ✓ startdate

Table humanresources.employee
- ☐ birthdate
- ☐ businessentityid
- ☐ gender
- ☐ hiredate
- ☐ jobtitle
- ✓ loginid
- ☐ maritalstatus
- ☐ nationalidnumber

Adjust privileges for user db user

Figure 6: Privilege adaption form

several systems dedicated to query management, e. g. [2, 10]. However, these tools are aimed at experts and lack usability for privacy officers. During the query-log analysis of [3], we considered the query structure and detected similarities among queries. However, we did not yet provide a way to report user-specific meta-information about queries.

QueRIE [9] performs a query-log analysis to give personalized query recommendations. Manta and SQLDep[3] visualize schema lineage of SQL queries. None of these approaches considers query context and other meta-information. This is also the case for several graph-based approaches to representing SQL queries, e. g. [8, 11]. Tools like the Query Patroller [14] or the DataLawyer [13] provide a wide range of mechanisms for query analysis. However, they aim at SQL experts and the latter is even limited to a particular DBMS.

5 CONCLUSION AND FUTURE WORK

The DataEconomist supports several aspects of data minimization: Based on the query-driven approach, user access to unneeded schema elements can be automatically customized. Completely unneeded data can be automatically detected and deleted. These features minimize data processing. In first measurements, we determined the average latency for analyzing the structure (117ms) and detecting unneeded schema elements (19ms) of a single query. The structure analysis includes the generation of the query execution plan, the detection works with this execution plan. We intend to present more detailed measurement results in a follow-up publication.

Our framework also points at queries inserting unneeded data and at these queries' authors. That allows privacy officers to take specific measures in minimizing data collection.

Up to now, privacy officers have to manually classify illegal queries regarding data privacy, which can be an outrageous effort. Therefore, we work on a semi-automatic classification of illegal queries to minimize user interaction. We are also enhancing our framework by an intuitive visualization of all data processed by a query to fully support non-expert SQL users in classification. Furthermore, we aim to additionally consider unneeded tuples for a more fine-grained data minimization.

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

[3]https://getmanta.com/ and https://sqldep.com/

REFERENCES

[1] Council of European Union. 2016. Council regulation (EU) no 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *OJ* L 119 (4 5 2016), 1–88.
[2] Jan Van den Bussche et al. 2005. Towards practical meta-querying. *Inf. Syst.* 30, 4 (2005), 317–332.
[3] Andreas M. Wahl et al. 2018. A graph-based framework for analyzing SQL query logs. In *Proceedings of the 1st ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA), Houston, TX, USA, June 10, 2018.* 11:1–11:5.
[4] Christina Tikkinen-Piri et al. 2018. EU General Data Protection Regulation: Changes and implications for personal data collecting companies. *Computer Law & Security Review* 34, 1 (feb 2018), 134–153.
[5] Divesh Srivastava et al. 2007. Intensional associations between data and metadata. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007.* 401–412.
[6] Edmon Begoli et al. 2018. Apache Calcite: A Foundational Framework for Optimized Query Processing Over Heterogeneous Data Sources. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018.* 221–230.
[7] Fei Li et al. 2014. Constructing an Interactive Natural Language Interface for Relational Databases. *PVLDB* 8, 1 (2014), 73–84.
[8] George Papastefanatos et al. 2005. Hecataeus: A Framework for Representing SQL Constructs as Graphs. In *Proceedings of 10th International Workshop on Exploring Modeling Methods for Systems Analysis and Design-EMMSAD*, Vol. 5.
[9] Magdalini Eirinaki et al. 2014. QueRIE: Collaborative Database Exploration. *IEEE Trans. Knowl. Data Eng.* 26, 7 (2014), 1778–1790.
[10] Nodira Khoussainova et al. 2009. A Case for A Collaborative Query Management System. In *CIDR 2009, Fourth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2009, Online Proceedings.*
[11] Nodira Khoussainova et al. 2010. SnipSuggest: Context-Aware Autocompletion for SQL. *PVLDB* 4, 1 (2010), 22–33.
[12] Peter K. Schwab et al. 2018. Towards Query-Driven Data Minimization. In *Proceedings of the Conference "Lernen, Wissen, Daten, Analysen", LWDA 2018, Mannheim, Germany, August 22-24, 2018.* 335–338.
[13] Prasang Upadhyaya et al. 2015. Automatic Enforcement of Data Use Policies with DataLawyer. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015.* 213–225.
[14] Sam Lightstone et al. 2002. Toward Autonomic Computing with DB2 Universal Database. *SIGMOD Record* 31, 3 (2002), 55–61.
[15] Thibaud Antignac et al. 2014. Privacy Architectures: Reasoning about Data Minimisation and Integrity. In *Security and Trust Management - 10th International Workshop, STM 2014, Wroclaw, Poland, September 10-11, 2014. Proceedings.* 17–32.
[16] Michelle Goddard. 2017. The EU General Data Protection Regulation (GDPR): European Regulation that has a Global Impact. *International Journal of Market Research* 59, 6 (nov 2017), 703–705.
[17] Marc Langheinrich. 2001. Privacy by Design - Principles of Privacy-Aware Ubiquitous Systems. In *Ubicomp 2001: Ubiquitous Computing, Third International Conference Atlanta, Georgia, USA, September 30 - October 2, 2001, Proceedings.* 273–291.