Query-Driven Knowledge-Sharing for Data Integration and Collaborative Data Science

Andreas M. Wahl(✉), Gregor Endler, Peter K. Schwab, Sebastian Herbst, and Richard Lenz

Computer Science 6 (Data Management), FAU Erlangen-Nürnberg, Erlangen, Germany
andreas.wahl@fau.de

Abstract. Writing effective analytical queries requires data scientists to have in-depth knowledge of the existence, semantics, and usage context of data sources. Once gathered, such knowledge is informally shared within a specific team of data scientists, but usually is neither formalized nor shared with other teams. Potential synergies remain unused. We introduce our novel approach of *Query-driven Knowledge-Sharing Systems (QKSS)*. A *QKSS* extends a data management system with knowledge-sharing capabilities to facilitate user collaboration without altering data analysis workflows. Collective knowledge from the query log is extracted to support data source discovery and data integration. Knowledge is formalized to enable its sharing across data scientist teams.

1 Introduction

Data scientists work according to their expert knowledge gained by solving previous data analysis challenges and maintain individual mental models of the available data sources. These models encompass knowledge about when certain data sources are useful, how they can be linked, how their content can be interpreted or what domain vocabulary is used. Within a team of data scientists, this knowledge is shared through personal interaction. In most cases however, it is not formally documented or shared between teams. Due to the complexity of analytical questions, it is common that multiple teams of data scientists work separately on data analysis challenges, especially in larger organizations. When different teams do not directly interact with each other, they miss out on opportunities to share their knowledge and to profit from experiences of others.

Contribution. To overcome the above deficiencies, we propose *Query-driven Knowledge-Sharing Systems (QKSS)*. A QKSS extends a data management system by adding services that formalize knowledge implicitly contained in a centralized query log and make it available to other users. Parts of the underlying mental model of each query are extracted. This model is mapped to actually available data sources by using previously generated mappings of related queries.

Our contribution comprises multiple aspects: **(1)** We introduce *shards of knowledge* as an abstraction for the concepts behind query-driven knowledge-sharing. **(2)** We provide a formal model for building, evolving, and querying shards. **(3)** We explain the integration of a QKSS with existing data analysis tools and processes by suggesting a reference architecture.

2 Query-Driven Knowledge-Sharing Systems

We consider the term *knowledge* to denote domain knowledge about data sources required to query them. Such knowledge contains, among others, the following aspects:

(1) What data sources are available? **(2)** What parts of data sources can be used for specific analytical purposes? **(3)** Which vocabulary and semantics are used to describe data sources? **(4)** How can data sources be related to each other? **(5)** Who is using which data sources in which temporal context?

2.1 Services of a QKSS

To explain the services a QKSS offers to data scientists and the benefits it provides, we describe a user story from a clinical research scenario, including simplified example queries.

Consider three teams of data scientists accessing a QKSS (Fig. 1). The QKSS manages different data sources containing data from electronic health records. Alice and Bob from team 1 use medication plans from data source D1 (JSON format) and the relational database D2 for their main focus of drug dosage analysis. Team 2 (Carol and Dan) also conduct drug dosage analysis, but rely on data from the relational databases D2, D3 and data source D4 (CSV format). Erin constitutes team 3 and specializes in time series analysis of patient monitoring data. She uses two data sources D5 (Avro format) and D6 (Parquet format) from a distributed file system.

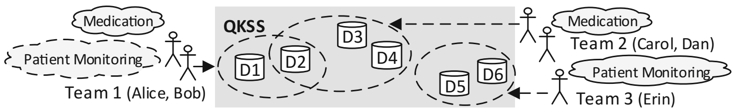

Fig. 1. Collaboration through a QKSS

Shared-Knowledge Support for Querying. Initially, none of the teams is aware of the others. The QKSS provides a SQL interface for data access. While Alice from team 1 is waiting for the completion of a query (Fig. 2), the QKSS detects that both teams rely on D2 by analyzing previous queries (Fig. 3).

The QKSS subsequently presents Alice with information and hints for future queries based on the collective knowledge of team 2. By analyzing queries of

```
SELECT D2.id, D2.Department
FROM D1 JOIN D2 ON D2.id = D1.PatNr
WHERE D1.Agent LIKE 'Dexametha%';
```

Fig. 2. Exemplary query by Alice

```
SELECT MIN(D2.Age)
FROM D4 JOIN D2 USING id
WHERE D4.ActiveAgent = 'Salbutamol';
-------------------------------------
SELECT D2.id, D3.Substance, D3.Dose
FROM D3 JOIN D2 ON D2.id = D3.Patient
```

Fig. 3. Exemplary queries by Team 2

team 2, the QKSS finds that data sources D3 and D4 have already been linked to D2. It therefore shows Alice a unified view of D2 and these sources. To help Alice with exploring the newly discovered sources, the QKSS ranks them according to frequency of use and temporal occurrence within query sessions of team 2.

Bob has recently received the order to investigate whether the ingestion of certain active agent combinations correlates with the occurrence of critical vital parameters. He is not aware of any data sources containing vital parameters yet, but has a notion of the kind of data he is looking for. Bob imagines vital parameter entries to have attributes such as `HeartRate` and `BloodPressure`. He uses this mental model to formulate a query referencing the hypothetical data source `VitalParameters` (Fig. 4).

```
SELECT HeartRate, BloodPressure
FROM VitalParemeters
WHERE HeartRate >= 130;
```

Fig. 4. Exemplary query by Bob

```
SELECT BloodPr AS BloodPressure
FROM D5
WHERE HeartRate < 90 AND Time > '16-01-10';
--------------------------------------------
SELECT HRt AS HeartRate, BP AS
BloodPressure
FROM D6
WHERE PId = 'P41' AND TStmp = '1475693932';
```

Fig. 5. Exemplary queries by Erin

The QKSS utilizes his assumptions about the structure and semantics of a fictional data source named `VitalParameters` to suggest actual data sources. Using the knowledge extracted from Erin's queries (Fig. 5), the QKSS detects similarities between `VitalParameters` and the data sources D5 and D6. These have been queried before using similar structural assumptions and vocabulary. Thus, the QKSS can recommend D5 and D6 as replacement for the fictional data source in Bob's query. Erin's alias names for schema elements automatically become part of an ontology. Bob can use this ontology for easier comprehension of the vocabulary used by other teams. He can provide feedback about the suggestions through an interactive dialog. The QKSS remembers the mapping between Bob's expectations and the actually available data sources and offers to automatically generate a view that corresponds to his mental model. Bob can directly use this view in future queries.

Management of Shared Knowledge. Whenever synergies between teams are discovered, data scientists may decide to incorporate the shared knowledge of others in the representation of their own mental models. The QKSS provides mechanisms to subscribe to the knowledge contained in queries of others and therefore enables collaborative pay-as-you-go data integration. As long as Bob is involved in patient monitoring data analysis, he subscribes to Erin's queries and augments the formalized representation of his own mental model with hers. He can automatically see the same unified view of the patient monitoring data sources that she does. When he is no longer interested in this topic, he may unsubscribe from her queries and return to his prior medication-centric view.

2.2 Shards of Knowledge

To implement this subscription process and other QKSS services, we introduce *shards of knowledge* as an abstraction for knowledge from the query log. A shard of knowledge captures the mental model of data sources a group of data scientists forms over a period of time. We use the expression *shard* because single mental models may be incomplete, while the combination of all mental models yields the overall organizational knowledge inferable from the query log. As depicted in Fig. 6, shards encapsulate knowledge models constructed from specific portions of the query log. Subsequently, we formally describe the lifecycle of shards to illustrate their usage by data scientists.

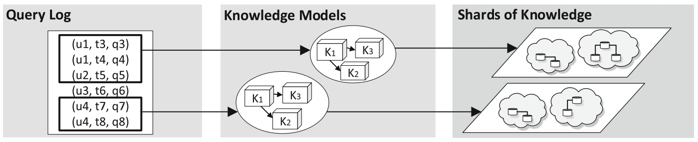

Fig. 6. From log entries to shards of knowledge

Instantiation. Shards are instantiated according to the preferences of data scientists. They determine which knowledge, in form of queries, is to be considered for each shard.

→ **Example:** An initial set of shards for a QKSS could encompass one shard for each team, incorporating all previous queries of the team members. In our example scenario, the QKSS manages an initial set of three shards.

Query Log. The query log L is a set of log entries of type $\mathcal{L}$. Each entry contains a user identifier u of type $\mathcal{U}$, a timestamp t of type $\mathcal{T}$ and a query q of type $\mathcal{Q}$ (Fig. 7(1)).

Only certain portions of the log L are relevant for the data scientists using the QKSS. These portions are extracted using functions `extr` that apply a set of filter predicates of type $\mathcal{F}$ on the log (Fig. 7(2)). We provide exemplary definitions suitable for a QKSS (Fig. 7(3)/(4)): Each filter predicate denotes the user u

whose queries are to be considered, as well as the time period of relevant log entries using the timestamps t_{start} and t_{end}. Using a set of filter predicates F of type $\mathcal{F}$, we can extract log entries from a log L using the function `extr`.

→ **Example:** For our scenario of three initial shards we would create the filter predicates F_1, F_2 and F_3 (Fig. 8). t_α and t_ω indicate that log entries from the whole lifespan of the QKSS are included and the QKSS considers future updates to the log. Fixed time spans could also be specified.

$$L : Set\ \mathcal{L} \quad \text{with } \mathcal{L} = (u : \mathcal{U},\ t : \mathcal{T},\ q : \mathcal{Q}) \tag{1}$$
$$\texttt{extr} : Set\ \mathcal{F} \times Set\ \mathcal{L} \rightarrow Set\ \mathcal{L} \tag{2}$$
$$\mathcal{F} = (u : \mathcal{U},\ t_{start} : \mathcal{T},\ t_{end} : \mathcal{T}) \tag{3}$$
$$\texttt{extr}(F, L) = \{l \in L | f \in F.((l.u = f.u) \wedge (f.t_{start} \leq l.t \leq f.t_{end}))\} \tag{4}$$

Fig. 7. Extracting relevant queries

$$F_1 = \{(Alice, t_\alpha, t_\omega), (Bob, t_\alpha, t_\omega)\}$$
$$F_2 = \{(Carol, t_\alpha, t_\omega), (Dan, t_\alpha, t_\omega)\}$$
$$F_3 = \{(Erin, t_\alpha, t_\omega)\}$$

Fig. 8. Exemplary filter predicates

Knowledge Models. Query log extracts are used by log mining algorithms to create knowledge models (Fig. 6). The algorithms a_i create individual data structures K_i from a subset of the query log to represent the extracted knowledge (Fig. 9(5)). The indices i are elements of an index set I that can be used to label all algorithms provided by the QKSS.

Each knowledge model of type $\mathcal{K}$ consists of the product of the different knowledge aspects extracted by the log mining algorithms (Fig. 9(6)). The function $\texttt{createModel}_{L,\texttt{extr},I}$ instantiates a model by applying the algorithms a_i on an extracted portion of the query log (Fig. 9(7)).

→ **Example:** For our example scenario, the QKSS provides a variety of log mining algorithms. An algorithm a_{links} extracts join edges between data sources from the query log to form a graph of data sources to reason over. Another algorithm $a_{session}$ partitions the query log into explorative sessions. An algorithm a_{struct} tracks how referenced portions of data sources are structured. To detect synonyms between the vocabularies of different teams, a_{onto} maintains an ontology. By using the resulting index set I (Fig. 10(a)), the knowledge model for team 1 is created (Fig. 10(b)), for example.

Shards. A shard of type $\mathcal{S}$ formalizes the mental model shared by multiple data scientists (Fig. 11(8)). A set F of filter predicates is used to instantiate the shard with the relevant portions of the query log. These are used to create a knowledge model K of type $\mathcal{K}$ which contains all knowledge of the shard.

The function $\texttt{createShard}_{L,\texttt{extr},I}$ takes a set of filter predicates F of type $\mathcal{F}$ to create a shard for a given Log L, a given extraction function `extr` and a given index set I of log mining algorithms (Fig. 11(9)). L, `extr` and I are identical for all shards of a specific QKSS instance.

$$a_i : Set\ \mathcal{L} \rightarrow K_i \quad \text{für } i \in I \tag{5}$$
$$\mathcal{K} = \prod_{i \in I} K_i = (K_{i_1}, ..., K_{i_n}) \tag{6}$$
$$\texttt{createModel}_{L,\text{extr},I} : Set\ \mathcal{F} \rightarrow \mathcal{K} \tag{7}$$
$$\texttt{createModel}_{L,\text{extr},I}(F) = \prod_{i \in I} a_i\ (\texttt{extr}(F, L))$$

Fig. 9. Creating knowledge models

$$I = \{links, session, struct, onto\} \tag{a}$$
$$\mathcal{K}_1 = \texttt{createModel}_{L,\text{extr},I}(F_1) = \tag{b}$$
$$= (a_{links}\ (\texttt{extr}(F_1, L)), a_{session}\ (\texttt{extr}(F_1, L)),$$
$$a_{struct}\ (\texttt{extr}(F_1, L)), a_{onto}\ (\texttt{extr}(F_1, L))) =$$
$$= (K_{links}, K_{session}, K_{struct}, K_{onto})$$

Fig. 10. Exemplary knowledge model

$$\mathcal{S} = (F : Set\ \mathcal{F},\ K : \mathcal{K}) \tag{8}$$
$$\texttt{createShard}_{L,\text{extr},I} : Set\ \mathcal{F} \rightarrow \mathcal{S} \tag{9}$$
$$\texttt{createShard}_{L,\text{extr},I}(F) = (F, \texttt{createModel}_{L,\text{extr},I}(F))$$

Fig. 11. Creating shards

$$s_1 = \texttt{createShard}_{L,\text{extr},I}(F_1)$$
$$s_2 = \texttt{createShard}_{L,\text{extr},I}(F_2)$$
$$s_3 = \texttt{createShard}_{L,\text{extr},I}(F_3)$$

Fig. 12. Exemplary shards

→ **Example:** In our scenario, each team might create a shard using the filter predicates F_1–F_3 (Fig. 12).

Evolution. Shards are dynamic and evolve over time. New log entries matching the filter predicates of a shard can become part of the underlying knowledge models. To provide data scientists with additional flexibility, the QKSS supports several operations to manage shards. While we consider shards to be immutable in our functional model, implementations may destroy or recycle shards.

Lifecycle Operations. Two shards can be merged into a single shard, to reflect in-depth collaboration between data scientist teams (Fig. 13(10)). Shards can also be expanded or narrowed (Fig. 13(11)/(12)). Thereby, specific parts of the query log can be added to or excluded from a shard.

$$\texttt{merge}_{L,\text{extr},I} : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{S} \quad \texttt{merge}_{L,\text{extr},I}(s1, s2) = \texttt{createShard}_{L,\text{extr},I}\ (s1.F \cup s2.F) \tag{10}$$
$$\texttt{expand}_{L,\text{extr},I} : \mathcal{S} \times \mathcal{F} \rightarrow \mathcal{S} \quad \texttt{expand}_{L,\text{extr},I}(s, F) = \texttt{createShard}_{L,\text{extr},I}\ (s.F \cup F) \tag{11}$$
$$\texttt{narrow}_{L,\text{extr},I} : \mathcal{S} \times \mathcal{F} \rightarrow \mathcal{S} \quad \texttt{narrow}_{L,\text{extr},I}(s, F) = \texttt{createShard}_{L,\text{extr},I}\ (s.F - F) \tag{12}$$

Fig. 13. Basic operations on shards

Shard Comparison. Two shards are considered similar if their knowledge models are similar. The knowledge extracted by log mining algorithms a_i can be compared by using algorithm-specific functions $\texttt{sim}K_i$ (Fig. 14(13)). Two knowledge models of type $\mathcal{K}$ can be compared to each other using a function $\texttt{sim}\mathcal{K}$ (Fig. 14(14)). A weight function w allows to determine the influence of specific

$$\texttt{sim}K_i : K_i \times K_i \rightarrow [0;1] \tag{13}$$

$$\texttt{sim}\mathcal{K} : (I \rightarrow [0;1]) \times \mathcal{K} \times \mathcal{K} \rightarrow [0;1] \tag{14}$$

$$\texttt{sim}\mathcal{K}(w, \mathcal{K}_1, \mathcal{K}_2) = \sum_{i \in I}(w(i) \cdot \texttt{sim}K_i(\mathcal{K}_1.K_i, \mathcal{K}_2.K_i)) \text{ with } \sum_{i \in I} w(i) = 1$$

$$\texttt{sim}\mathcal{S} : (I \rightarrow [0;1]) \times \mathcal{S} \times \mathcal{S} \rightarrow [0;1] \tag{15}$$

$$\texttt{sim}\mathcal{S}(w, s_1, s_2) = \texttt{sim}\mathcal{K}(w, s_1.K, s_2.K)$$

Fig. 14. Similarity functions

algorithms from the index set I on model similarity. The function $\texttt{sim}\mathcal{S}$ derives a numerical value for shard similarity by comparing the knowledge models of two shards (Fig. 14(15)). Data scientists can manually compare shards or rely on automatic comparisons by the QKSS. The system suggests evolution operations based on these comparisons, which can be reviewed by the data scientists.

→ **Example:** Assume that all algorithms except a_{struct} are assigned a weight factor of 0. Because of the structural similarity of s_1 and s_2 determined by $\texttt{sim}K_{struct}$ (s_2 incorporates exactly half of the sources of s_1), the QKSS automatically suggests team 1 to merge s_2 into s_1.

Query Processing. The knowledge model of a shard represents the mental model of the data scientists. Queries can be written against this mental model and do not have to reference actually available data sources. Before they can be evaluated against these data sources, the QKSS reasons over the knowledge model (Fig. 15). After logging a query, the QKSS determines if the query is fully specified, which means it only contains schema elements that belong to actually available data sources. If this is the case, the query is processed normally. The knowledge model belonging to the shard of the querying user is evaluated in parallel to generate recommendations. These may include hints about similar data sources, similar users, or synonyms for schema elements.

We also allow queries to be underspecified, as data scientists should be able to express queries using their mental models of the data sources. Whenever the QKSS encounters an underspecified query, it evaluates the knowledge model in order to modify the query to be processable using the actually available data sources. If it cannot decide about relevant data sources, it collects feedback from the user through an interactive dialog. Otherwise, the knowledge model is used to rewrite the query to be consistent with the mental model of the user while using actual data sources.

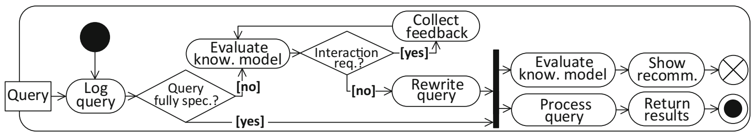

Fig. 15. QKSS query processing workflow

3 Reference Architecture

To demonstrate the feasibility of our approach, we are developing a reference QKSS [7]. This implementation adheres to our QKSS architecture (Fig. 16).

Users pose queries via the *SQL API* which is a wrapper for the native query interface of an existing data management system (DMS). Established analysis workflows remain intact, as analysis tools simply connect to the QKSS instead of a DMS. The *GUI* acts as a companion to present relevant knowledge and suggests modifications to the queries and shard lifecycle operations.

Incoming queries are stored in the centralized *query log* by the *query interception* component. Intercepted queries can be rewritten or extended and subsequently forwarded to the DMS. Result retrieval is handled by the DMS. The *shard management* oversees the lifecycle of all shards in the QKSS and generates knowledge models from the query log to be stored in the *model repository*. It also monitors the query log to update models if necessary. The *knowledge sharing* component provides relevant knowledge to rewrite or extend intercepted queries by evaluating the models from the model repository using the *inference engine*. It adjusts models according to the queries of the data scientists. Additionally, it monitors all shards of the QKSS to provide suggestions for lifecycle operations, such as merging similar shards.

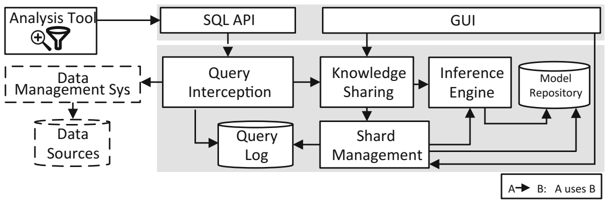

Fig. 16. QKSS reference architecture

4 Evaluation Methods

We assess the overall usefulness of our approach by analyzing how a QKSS supports data scientists with their data analysis tasks. Additionally, we examine if the performance of our reference implementation is sufficient for analytical ad-hoc queries and interactive usage.

Usefulness: The usefulness of our approach is evaluated during a user study. Knowledge models are created from queries of the participating users. The models are presented to the users to judge to what extent their intentions are correctly captured. By using logs of varying size, the number of queries required to create meaningful models can be assessed. To assess the result quality of shard comparisons, participating users validate if knowledge models that are marked similar by the QKSS are actually built from similar log portions. Subsequently, two groups of users are formed. Both groups get the task to answer specific analytical questions using a given set of unfamiliar data sources. While both groups use the QKSS to access the data sources through a single interface, only one group receives recommendations based on prepared knowledge models from the QKSS. We analyze how a QKSS can support the users by comparing the working speed and the result quality of both groups.

Performance: We measure the computational effort required for initial knowledge model creation, model maintenance when new queries are added to the log, model evaluation during query processing, and model comparison. To simulate different application environments, query logs of varying size and complexity are processed by the log mining algorithms. We also measure the overall response time of the system during the processing of ad-hoc queries.

5 Related Work

Dataspace systems [4] rely on user feedback to incrementally adapt the managed data to the expectations of their users. However, existing implementations of dataspace systems do not sufficiently consider scenarios where different groups of users with heterogeneous expectations work with a common set of data sources.

Mental models of data scientists may differ from the actual schema and content of data sources. Some approaches allow queries with references to unknown schema elements [2,6]. However, they do not consider advanced temporal and social connections inferable from the query log.

Our approach differs from query recommendation [3] and completion [5], as we want to enable the users to specify complete queries using individual mental models. We aim to adjust the actually available data sources to the mental models of the users and not to force users to adjust to the data sources. Thus, we minimize bias caused by anchoring and adjustment, psychological phenomena that have been found to have adverse effects on query and result quality [1].

6 Summary

We introduce *Query-driven Knowledge-Sharing Systems (QKSS)* to support data scientists in integrating these data sources and querying them for data analysis tasks. Using a QKSS, data scientists can externalize tacit knowledge about data sources without manual documentation effort, explore how others interact with data sources, and discover relevant data sources. *Shards of knowledge* provide an intuitive abstraction for the user-facing concepts of a QKSS. They encapsulate knowledge models derived from relevant portions of the query log.

References

1. Allen, G., Parsons, J.: Is query reuse potentially harmful? Anchoring and adjustment in adapting existing database queries. ISR **21**(1), 56–77 (2010)
2. Eberius, J., Thiele, M., Braunschweig, K., Lehner, W.: DrillBeyond: processing multi-result open world SQL queries. In: SSDBM 2015 (2015)
3. Eirinaki, M., Abraham, S., Polyzotis, N., Shaikh, N.: QueRIE: collaborative database exploration. KDE **26**(7), 1778–1790 (2014)
4. Franklin, M., Halevy, A., Maier, D.: From databases to dataspaces: a new abstraction for information management. SIGMOD Rec. **34**(4), 27–33 (2005)
5. Khoussainova, N., Kwon, Y., Balazinska, M., Suciu, D.: SnipSuggest: context-aware autocompletion for SQL. PVLDB **4**(1), 22–33 (2010)
6. Li, F., Pan, T., Jagadish, H.V.: Schema-free SQL. In: SIGMOD 2014 (2014)
7. Wahl, A.M.: A minimally-intrusive approach for query-driven data integration systems. In: ICDEW 2016 (2016)