Karlsruher Institut für Technologie

Karlsruhe Reports in Informatics 2019,3
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

On the Usefulness of SQL-Query-Similarity Measures to Find User Interests

Natalia Arzamasova, Klemens Böhm, Bertrand Goldman, Christian Saaler and Martin Schäler

2019

KIT – University of the State of Baden-Wuerttemberg and National Research Center of the Helmholtz Association

Fakultät für **Informatik**

On the Usefulness of SQL-Query-Similarity Measures to Find User Interests

Natalia Arzamasova, Klemens Böhm, Bertrand Goldman, Christian Saaler and Martin Schäler

Abstract— In the sciences and elsewhere, the use of relational databases has become ubiquitous. An important challenge is finding hot spots of user interests. In principle, one can discover user interests by clustering the queries in the query log. Such a clustering requires a notion of query similarity. This, in turn, raises the question of what features of SQL queries are meaningful. We have studied the query representations proposed in the literature and corresponding similarity functions and have identified shortcomings of all of them. To overcome these limitations, we propose new similarity functions for SQL queries. They rely on the so-called access area of a query and, more specifically, on the overlap and the closeness of the access areas. We have carried out experiments systematically to compare the various similarity functions described in this article. The first series of experiments measures the quality of clustering and compares it to a ground truth. In the second series, we focus on the query log from the well-known SkyServer database. Here, a domain expert has interpreted various clusters by hand. We conclude that clusters obtained with our new measures of similarity seem to be good indicators of user interests.

Index Terms— SQL log analysis, SQL query representations, similarity measures

1 Introduction

DATA with a specific structure is often stored in relational databases. This is the case both within companies as well as in more open settings such as sciences. In any case, such databases provide generic interfaces so that basically any information need can be formulated – and this is what individuals typically do. These information needs are manifold and depend on the user interests[1] and the background knowledge of users. For any organization or anybody providing database content, the question of what users find interesting is extremely important. In a scientific domain, a user interest may represent a research trend. In business, it may point to popular data slices, which one might want to refactor for better accessibility.

A promising way to find user interests is query-log analysis. An SQL query log provides an appropriate level of abstraction as well as precise information regarding the interests of users. Due to their declarative nature, SQL queries are relatively easy to interpret. To find areas of high interest in the data space, it is reasonable to cluster the SQL requests of a query log. This idea, however, comes with the following problems:

1. To cluster SQL queries, one needs a notion of query similarity. This leads to the question what meaningful features of SQL queries are, and how to extract them. Among others, we currently are aware of the following query representations:
    1. The feature-based (FB) representation[2] [1] focuses on the query structure.
    2. The witness-based (WB) representation [2] relies on the result of a query to a database.
    3. The access area-based (AAB) representation [3] captures the area of a data space that a user is interested in.

Example 1. Think of a query log consisting of the queries listed in Table 1. All three queries access table ‘Employees’. One might find the first and the second query similar. This is because they have the same structure, asking for employees in a particular department. Indeed, according to the FB approach, these two queries are identical. However, one can also disagree with this conclusion. In line with the WB representation, these two queries do not have any common tuples in their results sets. When it comes to AAB, the first and the second query refer to different parts of the data space and hence are not similar. Regarding the similarity of the first and the third query one cannot really say much. Even though there could be employees from the sales or the store department who started to work after 01/12/2015 (similarity in WB), this does not lead to meaningful insights. A user might have had different intentions when formulating these queries.

So far, to our knowledge, there is no comparative study on the usefulness of different query representations for clustering with the aim of finding user interest.

2. Having a query representation is not sufficient to cluster SQL queries. Based on this representation, it

TABLE 1
Queries in a log

#	Statements	Result
1	`SELECT * FROM Employees E WHERE E.department = 'sales'`	12 employees from sales department
2	`SELECT * FROM Employees E WHERE E.department = 'store'`	8 employees from store department
3	`SELECT * FROM Employees E WHERE E.startdate > '01/12/2015'`	10 employees who started working in a company after the date

[1] As defined in [Zeng et al., 2010], a user interest is the subject a user or a group of users wants to get to know. In this article, ‘user interest’ is an interest of many users.

[2] The name of the approach is one we have come up with, as with the approaches that follow.

is necessary to have a query-similarity function, quantifying for any two queries to what extent they are alike. The FB and WB representations lend themselves to straightforward overlap measures. The similarity function for AAB in turn proposed in [3] is complex compared to FB or WB. It also has some redundancies, and several definitions behind it are ad hoc, as we will explain. Generally speaking, we also wonder whether there are more answers to the question when two queries are similar.

3. Another challenge when clustering SQL queries is that we are not aware of any suitable publicly available data set including a ground truth. 'Suitable' means that it must include (1) a labeled SQL query log and (2) the database these queries have been submitted to. It also (3) must be publicly available.[3] So one cannot objectively compare similarity functions and the corresponding query representations.

This current paper studies the similarity of SQL queries through clustering of SQL query logs with the aim of identifying user interests within a data space. According to [4], a good clustering result must be *precise* and *interpretable*. These two criteria have guided us in our design considerations and the experiments. Our steps and the core insights are as follows:

1. We provide an extensive discussion of existing measures for query similarity and their advantages and disadvantages.
2. Based on this discussion, we propose a new kind of query similarity. It relies on the overlap and on the closeness of the access areas of the two queries. The existing notion of similarity based on access areas only takes the overlap into account. It also has some shortcomings, so we come up with a new access-area-based similarity. We also propose a new definition of overlap and will argue that it is more natural than the existing one.
3. We perform systematic experiments with the design alternatives. In particular, we study the impact of the various similarity functions and query representations on clustering quality.
4. To quantify the *precision* of clustering, data with a ground truth is needed. Having such data in our current domain is an issue that existing approaches apparently have difficulties with. We in turn come up with conditions where one knows in advance which cluster a query belongs to. Then we collect these queries together with this ground truth. We make this data publicly available.
5. To measure *interpretability* we conduct a study with an astronomer. He interprets various clustering results obtained from the SkyServer query log and assesses how well they align with user interests.
6. We find that our proposed similarity measures are better than the existing ones regarding both precision and interpretability and provide explanations for this. We have learned that the new measures are indeed helpful to arrive at 'query clusters' that are meaningful, i.e., represent user interests.

Paper outline: Section 2 introduces underlying notions. Section 3 reviews existing approaches to query log analysis, Section 4 features existing query representations and similarity functions. Section 5 covers our new similarity metrics, Section 6 experiments. Section 7 concludes.

2 PRELIMINARIES

We now introduce some underlying notions.

Definition 1. A relational database DB is a database consisting of N relations $R_1, \dots, R_N$.

Definition 2. The universal relation U of a query q is the Cartesian product of all relations occurring in the query q: $U = R_1 \times \dots \times R_N$.

To avoid clutter in the presentation, we assume that a relation occurs in a query at most once. Otherwise, the previous definition would need to be more complex.

Definition 3. A database schema S is a logical architecture of the database DB, i.e., a set of definitions of relations $R_1, \dots R_N$ of DB and constraints put on them.

Definition 4. A database state T of the database DB is data in the database DB allowed by the database schema S at any particular time.

Definition 5. A predicate P is a Boolean expression of all constrains put on U by a query q.

$(U;\ T)$ stands for the set of tuples of U at state T of DB.

Definition 6. Let a relational database DB be given. A query q is a Select-Project-Join (SPJ) request together with an optional aggregate computation.

In this paper, we only consider queries that fulfill Definition 6. We leave aside DML and DDL statements since they do not represent information needs.

Definition 7. A representation scheme QRS of a Query q is a function which returns certain feature values representing a query.

A query representation $QR(q)$ of a query q is a set of feature values which are the result of QRS applied to q.

Example 2. Think of a QRS which extracts the tables listed in the FROM clause, but nothing else. For the queries from Example 1,

$QR(q_1) = QR(q_2) = QR(q_3) = \{Employees\}$.

In contrast, if the QRS also extracts the attributes listed in the WHERE clause,

$QR(q_1) = QR(q_2) = \{Employees, Employees.department\}$

$QR(q_3) = \{Employees, Employees.startdate\}$.

Definition 8. A distance function $D(q_1, q_2)$ of Queries q_1 and q_2 is a function returning a nonnegative value.

Certain clustering algorithms impose conditions regarding the distance function used. So called semi-metric distances work with a broad variety of clustering algorithms. According to [5], such a distance $D(x_i, x_j)$ must satisfy the following conditions on a data set X:

1. Symmetry. $D(x_i, x_j) = D(x_j, x_i)$;
2. Positivity. $D(x_i, x_j) \geq 0$ for all x_i and x_j in X;

[3] We have considered two possible data sets: IIT Bombay [20] and the UB dataset[15]. The first data set however is not open access. The second one does not include the database. See Section 6.2.1.

3. Reflexivity. $D(x_i, x_j) = 0$ iff $x_i = x_j$;

We will check these conditions when introducing query-distance measures in Section 5.

Definition 9. The similarity $S(q_1, q_2)$ of two queries is a function returning a value in $[0;1]$.

$$S(q_1, q_2) = \begin{cases} 1 & if \quad q_1 = q_2 \\ [0;1) & \text{otherwise} \end{cases}$$

It holds that $S(q_1, q_2) = 1 - D(q_1, q_2)$.[4]

3 Related Work

Though we are about to study clustering of SQL query logs, the design of new clustering algorithms is not the focus of this article, and we do not provide a review of them here. This section also is relatively short since we defer the discussion of certain related work to the next section. There we will focus on the extraction of meaningful features out of SQL statements. A query presented as a set of features can relatively easily be compared to other ones, i.e., one can compute pairwise similarity values. Such similarities then are the input for clustering.

SQL query-log analysis allows solving specific issues with regard to database usage. One use case is the detection of performance problems. For instance, [6] scans query logs with the aim of finding patterns and antipatterns. It also claims that such antipatterns heavily influence subsequent analysis, like clustering or association-rule mining. The article provides some evidence, but does not feature a comprehensive evaluation. Related to the detection of antipatterns is [7], where researchers work with a log of update statements and a set of known data errors to find and fix mistakes in a dataset. But analyzing DML statements, as is done there, is not our current focus.

Another research thread applies association-rule mining to a query log [2] describes an approach that generates SQL query recommendations online. It compares user sessions and recommends a query to a user based on queries from similar sessions. The authors present the idea that similar user behavior manifests itself in similar data these users access. A different approach to similarity of SQL user sessions is presented in [8]. The paper focuses on OLAP sessions and introduces an order-sensitive model to compare them. This means that the order of queries within a session influences the similarity of sessions. The proposed method considers filtering conditions of queries in a limited way: Only equality predicates are allowed. Other related work [1] aims at autocompletion of a query, suggesting tables, views, UDFs, columns and predicates. It adjusts its recommendation to the context: The more of the query the user has typed in, the more accurate is the suggestion provided. [9] recommends join queries based on log analysis. They first extract chains of joins with corresponding predicates from the training set. The algorithm then creates queries from a test set with only tables present in these queries as an input.

A third research area, clustering SQL queries to identify hot spots of users' interests, is studied in [3]. It proposes a query-similarity metric based on the notion of so-called access areas. We discuss this notion in detail when reviewing query representations and similarity measures in the next section. [10] uses query clustering to help users locate interesting results. It generates clusters over the data. Each cluster corresponds to one type of user preference. In order to perform clustering, the authors compare queries based on the results they return. As an outcome, they present a navigational tree over clusters generated in the first step to the user. He can now select the subset of clusters matching his needs.

4 An Overview of Similarity Functions and Query Representations

In this section, we address the question how to define the similarity of queries. Since an SQL query may have a complex structure, this is not trivial. First one has to decide what to compare, i.e., which query-representation scheme (QRS) to use. We provide an overview of approaches we have encountered in the scientific literature. The Appendix summarizes the query representations reviewed so far and the corresponding similarity/distance functions. The last two rows refer to the new AAB similarity functions we are about to propose.

4.1 Query as a String

Arguably, the most straightforward way to represent an SQL query statement is as a string. To calculate query similarity, one could use string-similarity measures [11]. However, this hardly captures any specific features of SQL. We now elaborate on its drawbacks.

Example 3. Consider the following queries:

```
q1: SELECT * FROM Employees WHERE birthyear < 1980
q2: SELECT * FROM Employers WHERE birthyear < 1980
```

On the level of string similarities, these two queries have a very small difference – only one character. However, they access entirely different tables and therefore probably should have a very small similarity.

In addition, SQL keywords (SELECT, FROM, WHERE, etc.) overstate the similarity since they occur in every SPJ request. A possible solution is to exclude such words from consideration. However, this does not do away with effects like the one from Example 3 – even without keywords these two strings differ by only one symbol.

4.2 Query as a Set of Features

To overcome some of the obstacles described in Section 4.1 and to give more attention to the structure of an SQL request, [1] proposes a query representation as follows. There, a query is a set of features, and features are:

1. tables, views, UDFs in the FROM clause
2. attributes in the SELECT clause
3. predicates (without values) in the WHERE clause
4. attributes in the GROUP BY clause.

We refer to this query representation as the feature-based approach (FB). [1] is about autocompletion for SQL, not about clustering, and the authors do not explicitly

[4] In general there is no restriction on D, i.e., $D \in [0;\infty)$. This requirement is there exclusively for our query-distance function. We have introduced it in order to be able to set a meaningful threshold for the DBSCAN algorithm, to give an example.

present any notion of query similarity. However, based on the fact that a certain characteristic can only be present in a query or not, a feature can be seen as a binary attribute. Hence, one can use measures for binary attributes, such as the Jaccard coefficient, Sokal and Sneath, Gower and Legendre measures to arrive at similarity values for a pair of queries. For instance, using the Jaccard coefficient, the similarity of two queries q_1 and q_2 is:

$$S(q_1, q_2) = \frac{|features(q_1) \cap features(q_2)|}{|features(q_1) \cup features(q_2)|} \quad (1)$$

There are a few variants of FB: [12] aims at detecting anomalous access patterns in relational databases. It introduces three FB representations, which differ in details. [13] works with the SkyServer log and applies text mining techniques to parse, clean and tokenize statements into a weighted numerical representation, which can then be fed into regular machine learning. [14] represents an SQL query as an abstract syntax tree (AST) of its template. Any concrete values of such an AST are replaced by placeholders. To compare queries, the AST is then transformed to a vector of features. [15] focuses on how the feature-selection strategy affects clustering quality. It is confined to a feature-based query representation.

The FB approach captures the structure of the query and does not have the disadvantages of the method described first (Section 4.1). The main weakness of such a query representation is that it does not consider the values in a filtering condition.

4.3 Query as a Set of Result Tuples

Another query representation scheme [2], [16] introduces the notion of witnesses and is called witness based (WB) approach. A witness is a tuple in the result set of a query. A query representation is the set of its witnesses. The authors propose to measure the similarity of user sessions. A user session is a sequence of queries issued by one user. When generating query recommendations, they are interested in session-wise similarity, not query-wise. The similarity of two user sessions US_1 and US_2 is defined as cosine similarity. However, any metric for sets could be applied. For example, here is for the Jaccard coefficient:

$$S(US_1, US_2) = \frac{|witnesses(US_1) \cap witnesses(US_2)|}{|witnesses(US_1) \cup witnesses(US_2)|}$$

where $witnesses(US_i)$ is the set of witnesses which belong to user session US_i. Since a user session consists of queries, one can define query similarity in the same spirit:

$$S(q_1, q_2) = \frac{|witnesses(q_1) \cap witnesses(q_2)|}{|witnesses(q_1) \cup witnesses(q_2)|} \quad (2)$$

While this notion is very clear, we see several issues with this approach, as follows:

1. *Necessity to re-query the database.* To identify all witnesses, one must run the queries another time, leading to a huge load on the database. Next, even if this was not an issue, it spoils subsequent query-log analysis. This is because re-run queries are stored in the query log. Finally, due to possible updates of the database in the meantime, there is no guarantee that a query will have the same result as the first time.
2. *Result set can be empty*. Two queries which do not return any data cannot be compared even though they may be identical.
3. *Possible insignificance of witness sets*. Due to the declarative nature of SQL in particular, the same data can be obtained in many different ways. Consider again Example 1. It is possible for Queries q_1 and q_3 to have similar result sets. However, the intentions behind the two queries obviously are different.

Summing up, the WB approach overcomes the disadvantages of FB. It is clear and easy to implement. However, it may not be exactly practical in particular when the number of queries is very large.

4.4 Query as an Access Area

A way to overcome the disadvantages of the witness-based approach is proposed in [3]. The authors represent a query using the notion of so-called access areas. From now on, AAB is short for ‘access area based query representation’. The access area of a query captures the area of the data space that the user is interested in.

Definition 10. A tuple $t \in U$ is said to influence the result set $(U,T)P$ of a query q iff $(U\backslash\{t\},T)P \neq (U,T)P$. If t is removed from U, the result set of q at state T will change.

Definition 11. The access area of a query q is the set of all tuples t contained in the universal relation U that influence the result set of q in some database state T allowed by the database schema which satisfy Predicate P of conditions put on a query q:

$$\{t \in U : \exists T \; allowed \; by \; DB \; s.t. \, t \; influences \; (U,T)P\} \quad (3)$$

In contrast to WB, the access area of a query does not rely on the current database state. In many cases, we can describe these tuples as an expression in the relational algebra. Coming back to q_1 from Example 1, the access area of Query q_1 is $\sigma_{department = \text{'sales'}}(Employees)$. The notion leaves aside the SELECT clause of the query. It considers predicates and the FROM clause. These similarities do not consider attributes in a SELECT clause either. [1] describes how to compute access areas for simple queries as well as for join, aggregate and nested queries. Moreover, it proposes a query distance measure, based on the overlap of the access areas of the two queries:

$$D(q_1, q_2) = d_{tables}(q_1.FROM, q_2.FROM) + d_{conj}(q_1.WHERE, q_2.WHERE) \quad (4)$$

$$d_{tables}(q_1.FROM, q_2.FROM) = 1 - \frac{|q_1.FROM \cap q_2.FROM|}{|q_1.FROM \cup q_2.FROM|} \quad (5)$$

The predicate P is in conjunctive normal form (CNF), i.e., it is a conjunction of clauses, where each clause is a disjunction of literals. Hence, $d_{conj}(b_1; b_2)$ in Formula (4) means distance of conjunctions. It is calculated as:

$$d_{conj}(b_1; b_2) = \frac{\sum_{o_1 \in b_1} \min_{o_2 \in b_2} d_{disj}(o_1; o_2) + \sum_{o_2 \in b_2} \min_{o_1 \in b_1} d_{disj}(o_1; o_2)}{|b_1| + |b_2|}$$

where each $o_i \in b_i$ is a disjunction of Boolean expression(s), and $|b_i|$ is the number of disjunctions of b_i in Query q_i. $d_{disj}(o_1; o_2)$ is the distance of the disjunctions of

o_1 and o_2. It is as follows:

$$d_{disj}(o_1;o_2) = \frac{\sum_{p_1 \in o_1} \min_{p_2 \in o_2} d_{pred}(p_1;p_2) + \sum_{p_2 \in o_2} \min_{p_1 \in o_1} d_{pred}(p_1;p_2)}{|o_1| + |o_2|}$$

where $p_1 \in o_1$ is an atomic predicate, and $|o_1|$ is the number of atomic predicates of o_1.

The distances between predicates are:

- $d_{pred}(P_1,P_2) = 1 - \frac{overlap(a)}{width(a)}$ if both predicates P_1 and P_2 refer to the same attribute a;
- $d_{pred}(P_1,P_2) = 1 - \frac{interval(a_1)}{width(a_1)} \times \frac{interval(a_2)}{width(a_2)}$ if both predicates P_1 and P_2 refer to different attributes a_1 and a_2.

$interval(a)$ is the width of the interval in which Predicate P is true. The Appendix contains an example illustrating the notions which we have just defined. While the notion of access area itself has proven its worth, the distance function has several shortcomings:

1. The distance function is redundant, as follows: Formula (4) sums up the distance of the access tables, calculated using the Jaccard coefficient, as well as the distance of the conjunctions in the filtering conditions. If two queries have the same attributes in the filtering conditions, they have common tables in the FROM clause as well. Taking the distance of the tables accessed when one already calculates a distance of the filtering conditions is redundant. The following example illustrates this.

Example 4. Think of a query log containing the queries:

```
q1: SELECT * FROM Cities C WHERE C.latitude BETWEEN 30
and 50
q2: SELECT * FROM Cities C, Countries Cs WHERE
C.latitude BETWEEN 40 and 60 AND C.countryId = Cs.id
```

The access areas of these queries are:

q_1: $\sigma_{Cities.latitude \geq 30 \wedge Cities.latitude \leq 50}(Cities)$;

q_2: $\sigma_{Cities.latitude \geq 40 \wedge Cities.latitude \leq 60}(Cities \times Countries)$.

The first addend of the distance measure is as follows:

$d_{tables}(q_1.FROM, q_2.FROM) = 1 - \frac{|\{Cities\}|}{|\{Cities,Countries\}|} = \frac{1}{2}$.

To calculate $d_{conj}(q_1.WHERE, q_2.WHERE)$, the authors rely on the domain of a column. For attribute $latitude$ in Example 4, $dom(Cities.latitude) = [-90;90]$. Hence the width of this attribute is:

$width(Cities.latitude) = |90 - (-90)| = 180$.

With predicates in the two queries referring to the same single column,

$d_{conj}(q_1.WHERE, q_2.WHERE) = 1 - (overlap\ (latitude))/ width(latitude) = 1 - 10/180 = 17/18$.

Thus, the overall distance in this example is even more than 1: $D(q_1,q_2) = 1/2 + 17/18 = 13/9$.

Because two distances are summed up, the result may be an overall distance greater than 1, while this ought to be the value indicating maximally dissimilar queries. Summing up values with different meanings/ with different units of measure does not yield results with a clear meaning. One might argue that addends show the degree of dissimilarity – this is their common unit. Then this degree should have at least the same range. However, as Example 4 has shown, this is not true: $d_{tables} \in [0;1]$, $d_{conj} \in [0;\infty]$.

2. The distance of two queries depends on the width of the attributes, see Example 4. Hence one cannot come up with a maximum distance in advance. This renders the choice of threshold values for clustering algorithms like DBSCAN difficult.
3. The similarity function presented in the paper is not a semi-metric. To show this, we calculate the distance of two identical queries from Example 4:
$d_{tables}(q_1.FROM, q_1.FROM) = 0$.
$d_{conj}(q_1.WHERE, q_1.WHERE) = 1 - overlap(latitude)/ width(latitude) = 1 - 20/180 = 8/9$.
$D(q_1,q_2) = 0 + 8/9 = 8/9$, while the reflexivity condition requires that $D(q_1,q_1) = 0$.
4. The distance calculates the overlap of the access areas even if the two queries have different attributes in the filtering conditions. The following example illustrates that this may be problematic.

Example 5. Think of a query log containing the queries:

```
q1: SELECT * FROM T WHERE a = 1
q2: SELECT * FROM T WHERE b = 2
```

In this case, the authors propose to set $d_{conj}(q_1.WHERE, q_2.WHERE)$ to the share of the joint space of the columns involved occupied by $q_1.WHERE$ and $q_2.WHERE$. So these two queries might end up in the same cluster. The clusters then might become too big and consist of disjoint areas of the data space.

In our opinion, these shortcomings impact the identification of user interests based on clusters severely. Namely, when a cluster represents several user interests, one cannot distinguish between them.

4.5 Summary

Table 2 shows the FB, WB and AAB representations of Query q_1 from Example 1. FB is structure oriented, WB is data-oriented, AAB is somewhere in between, introducing access areas. Since an AAB representation is not a feature vector, one cannot use standard similarity measures, but an AAB similarity function is needed. To get the FB representation, one only needs the query. For WB in turn, the query and access to the data is needed. AAB does not need the entire data, only some statistical properties, like extreme values of an attribute.

5 OUR AAB SIMILARITY FUNCTION

We now face the necessity to come up with an AAB similarity measure which does not have these adverse characteristics. Representing a query as its access area seems promising. It captures key details of an SQL request and does not consider the current state of a database, in contrast to the WB approach. However, as we have pointed out in Section 4.4, a different query-similarity function is necessary. We now propose a new function which is still based on the notion of access areas but does not suffer from the shortcomings discussed in the previous section. To come up with it, we ask:

1. Which queries are similar?
2. How to quantify similarity of two queries in the following cases?
   a. There are several occurrences of an attribute in the filtering conditions in both queries.

b. There are different attributes in the filtering conditions, while the queries have at least one common attribute.
c. At least one query contains joins.

While this list does not cover SQL in full, it does cover a superset of SPJ queries with possibly aggregation, cf. Section 2. – From the study of the SkyServer query log, the only one freely available, it turns out that there are two types of queries that require further discussion:

- *Queries with arithmetic operations in predicates*. Queries with arithmetic operations in predicates are difficult in general, for instance for DBMS query optimizers. The current version of our AAB approach does not cover these queries. However, to include queries with arithmetic operations into the consideration, one could resort to the WB approach. This is the case particularly since one of our similarity functions, which calculates overlap of access areas (dubbed AABovl in the following), and WB yield similar results, as we will show in Section 6.
- *Queries without a filtering condition*. This kind of query is useful in combination with a TOP-n clause. These queries often are the first queries a user might issue, with the aim of testing the database. In this case, they have relatively little to do with user interests. These queries also blur the aggregated access area of a cluster they belong to. In our case study with SkyServer, only 2.7 % of the queries are of this kind. So we have consciously decided to leave them aside in this current study.

This means that, when it comes to a real-world query log, the three cases (2.a, 2.b, 2.c) cover most of the queries actually occurring. We now present a new formulation of query similarity. A dissimilarity or distance function must meet the conditions from Definition 8. We will prove that our distance/ similarity function has these characteristics. Before doing so, we introduce some underlying notions.

Definition 12. The similarity measure $S(q_1, q_2).a$ of an attribute a of two queries q_1, q_2 is a similarity measure of queries q_1 and q_2 which is defined if the queries both have at least one filtering condition with Attribute a and is undefined otherwise.

For a query pair q_1 and q_2, there can be one or several conditions on Attribute a.

$D(q_1, q_2).a$ is the corresponding distance and is calculated as follows: $D(q_1, q_2).a = 1 - S(q_1, q_2).a$.

Definition 13. An ordinal attribute (OA) is one whose values have a natural order.

The values of such an attribute may or may not be from a domain that is continuous.

Definition 14. A nominal attribute (NA) is one whose values do not have a natural order.

Definition 15. The interest $Ints(q)$ of a query q is the set of attributes occurring in filtering conditions of q.

Example 6. Back to Example 1, $Ints(q_1) = Ints(q_2) = \{Employees.department\}$.

Definition 16. The common interest $comInts(q_1, q_2) = Ints(q_1) \cap Ints(q_2)$ of two queries q_1, q_2 is the set of interests which occur in both queries q_1 and q_2.

TABLE 2
Query representations of q_1 from Example 1

Method	Query representation
FB [1]	{*, Employees, Department}
WB [2]	{(4352, John, Doe), (4322, Mary, Smith), (4152, Ivan, Green), (4357, Sarah, Bing), (8352, Eva, Dallas), (4052, Stephen, Li),…, (4356, Boris, Johnson), (4322, David, Black)}
AAB [3]	$\sigma_{department = 'sales'}(Employees)$

Definition 17. The exclusive interest $exclInts(q_1, q_2) = \{Ints(q_1) \cup Ints(q_2)\} \setminus comInts(q_1, q_2)$ of two queries q_1, q_2 is the set of interests which occur in only one query.

In what follows, we use the notation P_i^a for a predicate occurring within Query q_i and referring to Attribute a. In general, several terms may represent the same predicate: For instance, if a predicate referring to Attribute a occurs in both Query q_i and q_j, then both P_i^a and P_j^a are in order. Likewise, if the predicate refers to Attributes a_1 and a_2, both $P_i^{a_1}$ and $P_i^{a_2}$ may stand for the predicate.

Definition 18. A set of intervals $A_i^{OA} = \{a_{i.1}^{OA}, \ldots, a_{i.k}^{OA}\}$ of a query q_i is one formed by predicate P with $OA\,a$.

Example 7. Consider Queries $q_1$and q_2:

```
q1: SELECT * FROM Cities C WHERE C.latitude BETWEEN 10
and 20 OR C.latitude BETWEEN 40 and 50
q2: SELECT * FROM Cities C WHERE C.latitude BETWEEN 10
and 20 OR C.latitude NOT BETWEEN 40 and 50
```

Query q_1 has the following set of intervals for ordinal attribute $Cities.latitude$: $Cities.latitude_1^{OA} = \{[10; 20], [40; 50]\}$.For Query q_2, $Cities.latitude_2^{OA} = \{[-90; 40], [50; 90]\}$, because $dom(Cities.latitude) = [-90; 90]$.

We now define how to extract intervals out of a query.

Definition 19. The set of intervals $A_i^{OA} = \{a_{i.1}^{OA}, \ldots, a_{i.k}^{OA}\}$ of a Predicate P_i^a associated with an ordinal Attribute a is as follows. If P_i^a is:

1. Atomic clause: A_i^{OA} is a singleton set containing exactly the clause;
2. Conjunction clause $(C_1 \land C_2)$:$A_i^{OA} = C_1 \cap C_2$; it is the intersection of the intervals in C_1 and C_2;
3. Disjunction clause $(C_1 \lor C_2)$: $A_i^{OA} = C_1 \cup C_2$; it is the union of intervals in C_1 and C_2;
4. Negative clause $(NOT\ C)$: $A_i^{OA} = {}^{-}C$; it is the inverse interval of C.

Definition 20. The width $width(a_{i.k})$ of interval $a_{i.k}^{OA}$ is defined as $width(a_{i.k}) = a_{i.k}^{max} - a_{i.k}^{min}$.

For instance, the queries from Example 5 have the following widths of intervals:
$width(a_{1.1}) = 20 - 10 = 10$; $width(a_{1.2}) = 50 - 40 = 10$;
$width(a_{2.1}) = 40 + 90 = 130$; $width(a_{2.2}) = 90 - 50 = 40$;

Definition 21. A set of values $A_i^{NA} = \{a_{i.1}^{NA}, \ldots, a_{i.k}^{NA}\}$ valid with regard to a Predicate P_i^a over a nominal attribute a is a set of values of a where each value satisfies the conditions put on a by the predicate P_i^a.

As for intervals, we introduce an extraction procedure:

Definition 22. The set of valid values of a PredicateP_i^a associated with a nominal Attribute a $A_i^{NA} = \{a_{i.1}^{NA}, \ldots, a_{i.k}^{NA}\}$ is as follows. If P_i^a is:

1. Atomic clause: A_i^{NA} is a singleton set containing exactly the clause;

2. Conjunction clause ($C_1 \wedge C_2$): A_i^{NA} is the intersection of valid values in C_1 and C_2;
3. Disjunction clause ($C_1 \vee C_2$): A_i^{NA} is the union of valid values in C_1 and C_2;
4. Negative clause ($NOT\ C$): A_i^{NA} is all the values from domain not presented in C.

5.1 Which Queries are Similar?

To come up with a measure of query similarity, we first study the simplest case, when two queries have one occurrence of the same attribute in the filtering condition and nothing else. It seems plausible that similar queries are those whose access areas overlap. However, this might be too strict in certain cases.

Example 8. Think of a query log containing the queries:

q_1: **SELECT** * **FROM** Cities C **WHERE** C.latitude **>=** 45 **AND** C.latitude **<** 90
q_2: **SELECT** * **FROM** Cities C **WHERE** C.latitude **>=** 30 **AND** C.latitude **<** 45
q_3: **SELECT** * **FROM** Cities C **WHERE** C.latitude **>=** -75 **AND** C.latitude **<** -30
q_4: **SELECT** * **FROM** Cities C **WHERE** C.name = 'New York'
q_5: **SELECT** * **FROM** Cities C **WHERE** C.name = 'Paris'

Attribute latitude of table Cities has a continuous type, $Cities.latitude \in [-90; 90]$. This attribute is ordinal (OA), while $Cities.name$ is nominal (NA). Now look at the first three queries in the log. Fig. 1 plots the access areas of Queries q_1, q_2 and q_3. They are as follows:

q_1: $\sigma_{latitude\ \geq 45\ \wedge\ latitude\ \leq 90}(Cities)$;
q_2: $\sigma_{latitude\ \geq 30\ \wedge\ latitude\ \leq 45}(Cities)$;
q_3: $\sigma_{latitude\ \geq -75\ \wedge\ latitude\ \leq -30}(Cities)$.

No two queries overlap. But q_1 and q_2 appear to be closer to each other: Their access areas even are adjacent.

So we need to take in closeness as a criterion as well. Observe that all already existing measures which rely on the data, like WB or AAB, currently do not feature this either. In other words, the phenomenon that closeness is neglected is not specific to access-area-based approaches.

5.1.1 *Closeness Similarity for Ordinal Attributes*

We want to quantify the closeness of the access areas of two queries. Lack of overlap of access areas does not mean 'zero similarity'. Put differently, queries which request data in neighboring parts of the data space should have the chance to end up in the same cluster.

Definition 23. The similarity of two queries with the same filtering ordinal attribute (OA) $S_{cl}(q_1, q_2).a$ is the proximity (closeness, cl) of their access areas:

$$S_{cl}(q_1,q_2).a = S_{cl}(a_{1.1}^{OA}, a_{2.1}^{OA}) = \frac{1}{2} \cdot \frac{\left(a_{1.1}^{max} - a_{1.1}^{min}\right) + \left(a_{2.1}^{max} - a_{2.1}^{min}\right)}{\max(a_{1.1}^{max}, a_{2.1}^{max}) - \min\left(a_{1.1}^{min}, a_{2.1}^{min}\right)} \quad (6)$$

$a_{i.1}^{min}$/ $a_{i.1}^{max}$ are the minimum/ maximum of Interval $a_{i.1}^{OA}$.

Since we are considering the simplest case, there is only one interval of one attribute for each query. Because of this, the similarity of attribute conditions is the similarity of the first occurrence of Attribute a in both queries: $S_{cl}(q_1,q_2).a = S_{cl}(a_{1.1}^{OA}, a_{2.1}^{OA})$. The coefficient 0.5 normalizes the measure. The formula is the share of the space accessed over the width of the space between the queries. $S_{cl} > 0.5$ indicates overlap of access areas.

LEMMA 5.1. $D_{cl}(q_1, q_2).a$ is semi-metric.

All proofs are in the Appendix.

5.1.2 *Overlap Similarity for Nominal Attributes*

The closeness measure $S_{cl}(q_1, q_2).a$ in Equation (6) does not work with nominal attributes (NA). See Queries q_4 and q_5 from Example 8. The values of Attribute $name$ of Table $Cities$ do not have a natural order. To illustrate, 'Paris' is not close to 'Prague' just because they both start with 'P'[5]. Having said this, for nominal attributes we propose to take the overlap (ovl) as the similarity. We use the Jaccard coefficient to this end:

$$S_{ovl}(q_1,q_2).a = \frac{|A_1^{NA} \cap A_2^{NA}|}{|A_1^{NA} \cup A_2^{NA}|} \quad (7)$$

In our case,

$$S_{ovl}(q_4,q_5).Cities.name = \frac{|\{\text{'New York'}\} \cap \{\text{'Paris'}\}|}{|\{\text{'New York'}\} \cup \{\text{'Paris'}\}|} = 0.$$

LEMMA 5.2. $D_{ovl}(q_1, q_2).a$, where a is a nominal attribute, is semi-metric.

5.1.3 *Overlap Similarity for Ordinal Attributes (OA).*

With the definitions so far, we would rely on different paradigms, i.e., closeness and overlap, when calculating the similarity for ordinal and nominal attributes. When different types of attributes are treated differently, it is unclear how this will affect analysis results, e.g., clustering. Therefore, to have an alternative which we can use as a reference point later, we now propose a purely overlap-based similarity measure for ordinal attributes:

$$S_{ovl}(q_1,q_2).a = S_{ovl}(a_{1.1}^{OA}, a_{2.1}^{OA}) = \frac{comWidth(a_{1.1}^{OA}, a_{2.1}^{OA})}{allWidth(a_{1.1}^{OA}, a_{2.1}^{OA})} \quad (8)$$

In each query, one interval $a_{1.i}^{OA}$ represents Attribute a. $comWidth(a_{1.1}^{OA}, a_{2.1}^{OA})$ in Formula (8) is the overlap of Queries q_1 and q_2 for Attribute a in absolute terms:

$$comWidth(a_{1.1}^{OA}, a_{2.1}^{OA}) = \max(0, (\min(a_{1.1}^{max}, a_{2.1}^{max}) - \max(a_{1.1}^{min}, a_{2.1}^{min})) \quad (9)$$

$allWidth(a_{1.1}^{OA}, a_{2.1}^{OA})$ is the difference between the highest maximal bound and the lowest minimal bound:

$$allWidth(a_{1.1}^{OA}, a_{2.1}^{OA}) = \max(a_{1.1}^{max}, a_{2.1}^{max}) - \min(a_{1.1}^{min}, a_{2.1}^{min}) \quad (10)$$

For instance, the similarity for Attribute $Cities.latitude$ from Example 4 is:

$$S_{ovl}(q_1,q_2).latitude = \frac{\min(50,60) - \max(30,40)}{\max(50,60) - \min(30,40)} = \frac{50-40}{60-30} = \frac{1}{3}.$$

Fig. 2 graphs the corresponding access areas.

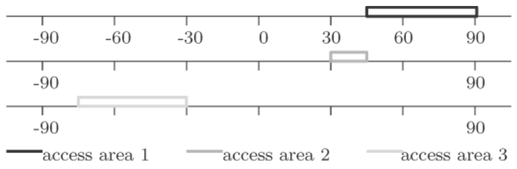

Fig. 1. Access areas of attribute Cities.latitude for queries q_1, q_2 and q_3 from Example 8.

[5] In principle, one can use domain-specific ontologies and respective distance measures. To continue the example, Paris and Prague might be similar because they both are capitals of European countries with a rich history. However, taking such additional information into account is beyond the scope of this paper.

LEMMA 5.3. $D_{ovl}(q_1, q_2).a$, where a is an ordinal attribute, is semi-metric.

5.1.4 *Summary*

We have identified two paradigms of AAB query similarity: closeness (AABcl) and overlap (AABovl). We use these acronyms from now on. We also have proposed query-similarity measures for ordinal attributes (Formula (6) for AABcl and (8) for AABovl) and nominal ones (Formula (7)). Which method to apply (closeness or overlap) when it comes to ordinal attributes depends on the objective. Our hypotheses, which our experimental evaluation will address, are as follows: If an analyst is interested in exact access areas many users have looked for, he might want to use the “strict” overlap formula (8). In contrast, if he is more interested in the bigger picture, i.e., approximate, rather big areas users have looked at, the less strict closeness formula (6) might be better. Fig. 3 shows the clustering results with the two approaches. Our experiments in Section 6 will provide more details.

So far, we have discussed similarity measures for the simplest case, two queries having the same attribute in the filtering conditions, and this attribute occurs only once. Now we turn to more complex cases.

5.2 Multiple Occurrences of an Attribute in Filtering Conditions

The first complication when calculating query similarity, described in the beginning of Section 5, occurs when one uses the same attribute several times in the same query. This may happen when a query consists of OR predicates (for ordinal and nominal attributes) or IN predicates (for nominal attributes).

5.2.1 *Overlap Similarity*

Example 9. Consider the following queries:

```
q1: SELECT * FROM Cities WHERE (population BETWEEN 0 and
20) OR (population BETWEEN 40 and 60)
OR (population BETWEEN 90 and 100)
q2: SELECT * FROM Cities WHERE (population BETWEEN 10
and 30) OR (population BETWEEN 50 and 60)
OR (population BETWEEN 80 and 90)
```

The access areas of the queries look like in Fig. 4. Both Queries q_1 and q_2 have not one, but several occurrences of Attribute *Cities.population*. Some of the areas do intersect: for instance, $(population\ BETWEEN\ 0\ and\ 20)$ of q_1 and $(population\ BETWEEN\ 10\ and\ 30)$ of q_2. Some however, like ($population$ BETWEEN 40 and 60) of q_1 and $(population\ BETWEEN\ 80\ and\ 90)$ of q_2, do not.

With an *overlap* approach for ordinal attributes, we define the similarity measure as the ratio of the width of the overall overlap of the intervals to the width of the union of the intervals. Formally,

$$S_{ovl}(q_1, q_2).a = \frac{ovlComWidth(a_1, a_2)}{ovlAllWidth(a_1, a_2)} \tag{11}$$

The terms in the numerator and in the denominator are as follows:

$$ovlComWidth(a_1, a_2) = \sum_{i=1, j=1}^{i=l_1, j=l_2} comWidth\left(a^{OA}_{1.i}, a^{OA}_{2.j}\right)$$

$$ovlAllWidth(a_1, a_2) = \sum_{i=1}^{i=l_1} width(a^{OA}_{1.i}) + \sum_{j=1}^{j=l_2} width(a^{OA}_{2.j}) - ovlComWidth(a_1, a_2)$$

where l_1 and l_2 are the numbers of intervals over Attribute a occurring in Queries q_1 and q_2 respectively. The width of an interval is calculated as in Definition 20.

Example 10. Think of the queries from Example 9. Here, $population_{1.1} = [0; 20], population_{1.2} = [40; 60]$, $population_{1.3} = [90; 100]$; $population_{2.1} = [10; 30]$, $population_{2.2} = [50; 60], population_{2.3} = [80; 90]$. $S_{ovl}(q_1, q_2).population = \frac{20}{50+40-20} = \frac{2}{7}$.

For nominal attributes, the formula remains exactly the same as in Section 5.1.2 (Formula (7)). This is because it already covers several occurrences of an attribute.

LEMMA 5.4. $D_{ovl}(q_1, q_2).a$, where a is an ordinal attribute which occurs several times in queries q_1 and q_2, is semi-metric.

5.2.2 *Closeness Similarity*

With the closeness similarity that we have considered so far, queries without overlap can be similar. Thus, Formula (11) is not applicable in this case. Hence, we propose to calculate overall closeness similarity for ordinal attributes (OA) as follows:

$$S_{cl}(q_1, q_2).a = \max_{i=1,\ldots,l_1; j=1,\ldots,l_2} S_{cl}(a^{OA}_{1.i}, a^{OA}_{2.j}) \tag{12}$$

The formula takes the maximum of pairwise similarities. So the closest intervals of two queries determine the similarity. *max* also has some desirable properties:

- It returns a normalized value in the [0;1] range of values. This is different from aggregation with, say, *sum*.
- It does not underestimate the similarity of two queries with the closeness paradigm. *min* or $\prod$ in turn do.

LEMMA 5.5. $D(q_1, q_2).a$ is semi-metric.

5.3 Several Distinct Attributes in Filtering Conditions

So far, we have discussed the cases when both queries filter with the same single attribute: $Ints(q_1) = Ints(q_2)$; $|Ints(q_1)| = |Ints(q_2)| = 1$. The following example illustrates the case of different attributes in the filtering conditions of two queries.

Example 11. A query log contains the following queries:

```
q1: SELECT * FROM Cities C WHERE C.latitude BETWEEN 52
and 80 AND C.longitude BETWEEN 30 and 45 AND
C.population BETWEEN 30 and 500
q2: SELECT * FROM Cities C WHERE C.latitude BETWEEN 40
and 52 AND C.longitude BETWEEN 30 and 45 AND C.country
='France'
```

Queries q_1 and q_2 have two common interests:

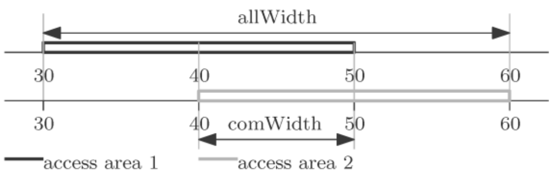

Fig. 2. Access areas of attribute *Cities.latitude* from Example 4

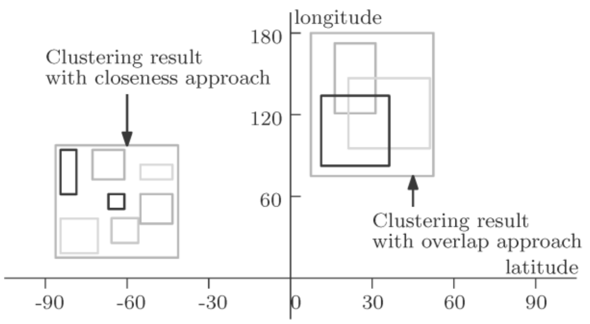

Fig. 3. Clustering result for closeness or overlap approach

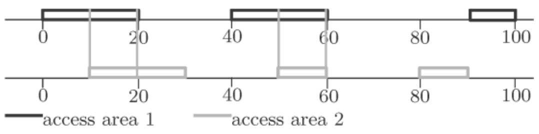

Fig. 4. Access areas of queries with multiple occurrence of an attribute from Example 9

$comInts(q_1, q_2) = \{C.latitude, C.longitude\}$.

They also have exclusive interests: $exclInts(q_1, q_2) = \{C.population, C.country\}$.

We propose to calculate the similarity measure $S_{attrFull}(q_1, q_2)$ where both $comInts(q_1, q_2)$ and $exclInts(q_1, q_2)$ are considered with similarities $S_{attrCom}(q_1, q_2)$ and $S_{attrExcl}(q_1, q_2)$. Section 5.3.3 will feature the concrete formula.

5.3.1 *Similarity for Common Interests*

So far, we have defined the attribute-wise similarity for queries $S(q_1, q_2).a$. If queries however have more than one common interest, as in Example 8, we need a definition which takes the attribute-wise similarities for all common attributes $S(q_1, q_2).a,\ a \epsilon\ comInts(q_1, q_2)$, as input. Since a query may contain ordinal and nominal attributes, we cannot separate closeness and overlap approaches. Instead, a general, unifying approach to arrive at meaningful overall similarities is necessary. Coming back to Example Example 11,

$S_{cl}(q_1, q_2).latitude = 0.5$; $S_{ovl}(q_1, q_2).latitude = 0$

$S_{cl}(q_1, q_2).longitude = 1$; $S_{ovl}(q_1, q_2).longitude = 1$

The overlap approach assumes that, if there is no overlap, then there is no similarity. Any non-overlapping condition should lead to zero similarity. For our example, $S_{ovl}(q_1, q_2) = 0$ since $S_{ovl}(q_1, q_2).latitude = 0$. In general,

$$S_{attrCom}(q_1, q_2) = \min_{i=1,\ldots,|comInts(q_1,q_2)|} S(q_1, q_2).a^i \qquad (13)$$

where a^i is an attribute contained in $comInts(q_1, q_2)$. min does not overestimate the similarity. Hence, one might expect relatively small clusters with clear user interests. Since we do not see any alternative how the overlap approach could be generalized, we use Formula (13) for the closeness approach as well.

LEMMA 5.6. $D_{attrCom}(q_1, q_2)$ is semi-metric.

5.3.2 *Similarity for Exclusive Interests*

If two queries have at least one shared interest, but also have exclusive ones, the similarity measure should reflect this. For each attribute in $exclInts(q_1, q_2)$, we calculate an overlap similarity value. We assume that an empty filtering condition in q_1 or q_2 means that one is interested in the entire domain of that attribute.

$$S_{attrExcl}(q_1, q_2) = \min_{i=1,\ldots,|exclInts(q_1,q_2)|} S_{ovl}(q_1, q_2).a^i \qquad (14)$$

Here, a^i is an attribute contained in $exclInts(q_1, q_2)$. In fact, $S_{attrExcl}(q_1, q_2)$ just calculates overlap similarity among attributes that do not occur in both queries.

Example 12. Let us now calculate $S_{attrExcl}(q_1, q_2)$ for queries from Example 11. Suppose that $C.country$ has 250 distinct values, and that $C.population \in [0; 20000]$.
$exclInts = \{C.country, C.population\}$;

$S_{ovl}(q_1, q_2).country = \frac{|\{France\}|}{|\{Afghanistan, \ldots, Zimbabwe\}|} = 0.004$;

$S_{ovl}(q_1, q_2).population = \frac{500-30}{20000} = \frac{47}{2000} = 0.0235$;

$S_{attrExcl}(q_1, q_2) = \min(0.004, 0.0235) = 0.004$.

There are two reasons for using overlap-based similarity for $S_{attrExcl}$:

- A non-shared filtering attribute can be nominal. We do not see any reason why non-shared ordinal and nominal attributes should be treated differently.
- We believe that dissimilar interests stand for different user intentions. The similarity values should be low. The closeness approach for ordinal attributes might yield clusters of queries without similar interests.

LEMMA 5.7. $D_{attrExcl}(q_1, q_2)$ is semi-metric.

5.3.3 *Overall Attribute Similarity.*

Finally, the minimum of $S_{attrExcl}$ and $S_{attrCom}(q_1, q_2)$ is the overall similarity of attributes:

$$S_{attrFull}(q_1, q_2) = \min\left(S_{attrExcl}(q_1, q_2), S_{attrCom}(q_1, q_2)\right) \qquad (15)$$

LEMMA 5.8. $D_{attrFull}(q_1, q_2)$ is semi-metric.

All predicates including join predicates are processed when we compute attribute similarities (Formula (15)). While we mostly use one- or two-dimensional examples, the principle is independent from this number.

5.4 Similarity in the Presence of Joins

The last remaining difficulty regarding our AAB similarity function is what needs to be done in the presence of joins.

Example 13. Consider the following query log:

q_1:
```
SELECT * FROM Cities C INNER JOIN Objects O ON
C.objId = O.objId WHERE O.latitude BETWEEN 52 and 80
AND O.longitude BETWEEN 30 and 45
```
q_2:
```
SELECT * FROM PowerStations PS INNER JOIN Objects O
ON PS.objId = O.objId WHERE O.latitude BETWEEN 52 and 80
AND O.longitude BETWEEN 30 and 45
```
q_3:
```
SELECT O.id FROM Objects O WHERE O.latitude BETWEEN
52 and 80 AND O.longitude BETWEEN 30 and 45
```
q_4:
```
SELECT O.id, T.typeName FROM Objects O INNER JOIN
Types T ON O.type = T.id WHERE O.latitude BETWEEN 52 and
80 AND O.longitude BETWEEN 30 and 45
```

Queries q_1 and q_2 look for different objects, i.e., entities from different relations, but in the same part of the data space. One must take this distinction into account. But our metric so far only relies on the filtering conditions.

An intuitive solution is to multiply the overall attribute-similarity values $S_{attrFull}(q_1, q_2)$ with a value quantifying the overlap of the sets of tables accessed, e.g., the Jaccard coefficient:

$$S_{final}(q_1,q_2) = \frac{|q_1.FROM \cap q_2.FROM|}{|q_1.FROM \cup q_2.FROM|} \cdot S_{attrFull}(q_1,q_2) \qquad (16)$$

where $q_i.FROM$ is the set of tables accessed by Query q_i.

This approach, while being simple, has a problem. Consider Queries q_3 and q_4. They search all objects within identical coordinate boundaries, i.e., intervals. q_4 has as additional output the type of an object which comes from the join with Table *Types*. According to Formula (16), the JOIN in Query q_4 decreases the similarity of q_3 and q_4 twice, from 1 to 0.5. Adding more joins to q_4, just to output more information, reduces similarity even more, compared to the query without joins. Hence, we argue that a more adequate reduction coefficient should consider the size of tables accessed, in rows:

$$S_{final}(q_1,q_2) = reductCoeff_{table} \cdot S_{attrFull}(q_1,q_2)$$

$$reductCoeff_{table} = \frac{\sum_{i=1}^{i=|T_{com}|} T_{com\,i}.size}{\sum_{j=1}^{j=|T_{all}|} T_{all\,j}.size} \qquad (17)$$

Here, T_{com} is the set of common tables of Queries q_1 and q_2, $T_{com} = q_1.FROM \cap q_2.FROM$. Accordingly, T_{all} is the set of all tables accessed in Queries q_1 and q_2, $T_{all} = q_1.FROM \cup q_2.FROM$.

Example 14. Let us suppose that $Types.size = 20$, $Objects.size = 980$. The reduction coefficient $reductCoeff_{table}$ for q_3 and q_4 is:
$T_{com} = q_3.FROM \cap q_4.FROM = \{Objects\}$;
$T_{all} = q_3.FROM \cup q_4.FROM = \{Types, Objects\}$.
$reductCoeff_{table} = \frac{Objects.size}{Cities.size+Objects.size} = \frac{980}{20+980} = \frac{49}{50}$.

$reductCoeff_{table}$ has a specific value for each query pair. One should apply it once after having calculated the overall attribute similarity $S_{attrFull}(q_1,q_2)$.

LEMMA 5.9. $D_{final}(q_1,q_2)$ is semi-metric.

Recall that the semi-metric characteristic is useful: It allows us to use our similarity/ distance function in many clustering algorithms without any further validation.

5.5 The Overall AAB Similarity Function

Summarizing what we have said so far, the overall AAB similarity function is as follows:

$$S(q_1,q_2) = reductCoeff_{table} \cdot S_{attrFull}(q_1,q_2) \qquad (18)$$

To calculate the similarity $S(q_1,q_2).a$ for an attribute which exists in both queries, one can use Formulas (11) or (12), depending on the approach, i.e., AABovl or AABcl. Consequently, Formulas (11) and (12)(12) refer to the simple case of Formulas (6), (7) and (8).

5.6 Discussion

In a nutshell, the AAB query representation captures parts of the data space where the user has an interest in. The WB query representation has the same objective, by identifying the relevant data explicitly. Hence, we expect to get similar results from clustering. However, as we have already pointed out, WB lacks scalability. AAB in turn does not have this limitation since it operates with access areas, not the data itself.

6 EXPERIMENTAL EVALUATION

This section evaluates various algorithms, query representations and similarity functions for query-log clustering. Our objectives are:

- Investigate the precision of the various QRSs (FB, WB, AAB) experimentally, on data where a ground truth is available. A QRS is *precise* if it leads to a clustering with a big overlap with the ground truth;
- Generate clustering results with real-world data, including the SkyServer query log in our case, inspect it and try to arrive at general insights;
- Study the influence of sampling on the clustering result.

6.1 Experiment Settings

The quality of any clustering is hard to evaluate without a ground truth. One can ask domain experts to provide an interpretation of the results. However, in our current context, a domain expert may not be able to say whether a result is good or even perfect – there often does not exist any expectation how an ideal result should look like. To cope with this problem to some extent, we propose two experiments. The first one is clustering queries which hundreds of individuals have formulated to solve a specific task. In our case, these individuals are university students participating in a database course. They have solved this task as part of an exercise where the information need was given in natural language. We have anonymized the data before our analysis so that it did not contain any personal information. This experiment 'only' serves to study the precision of the various query representations, by comparing clustering results to a ground truth. This is because the data set available is relatively small, and it has turned out that some query representations are more sensitive to it than another. Ideally, all queries that the students have formulated to solve a particular task should end up in one cluster.

The second experiment is a case study with a real-world SQL log. We use the SkyServer query log, a large log of queries on scientific data available to the public. To our knowledge it is the only query log publicly available. More specifically, we use the SkyServer log for 2016. It consists of 12.9 million queries from about 4,000 users.

For each data we first calculate pairwise distances, i.e., build a proximity matrix. There are well-established types of clustering methods which can work with a proximity matrix: hierarchical clustering, partitioning-based methods and density-based approaches. To make our study comprehensive, we select well-known instances from the different categories of clustering methods. We choose the hierarchical algorithm CLINK [17], k-medoids [18] from the class of partitioning-based methods and DBSCAN [19] as a density-based approach.

6.2 The Data Sets

In this section, we describe the data sets for our experiments – the small one from the student exercise and the big one from the SkyServer project.

6.2.1 *Data Set from the Student Exercise*

We have collected 1062 SQL requests formulated by 274 participants of the database course at our institution in the summer semester 2016. To facilitate repeatability, we make these queries and the test database publicly available[6]. We have asked the participants to produce solutions to four information needs. Thus, our ground truth is that all solutions to one information need form a cluster. Hence, we expect 4 clusters. Of course, not all answers have been complete and syntactically correct. Table 2 of the Appendix is a summary of GtDbCourseLog, the log with these queries from the database course.

We have considered using other labeled query logs. [20] however is not publicly available. [15] is the log of a database exam. First, it has only two tasks, i.e., one may expect two clusters. It is smaller than in our log. Second, the log is smaller in terms of the number of queries as well (178 against 1062). Moreover, [15] does not include the test database, which the WB query representation requires. So we have decided to use only one query log with a ground truth, GtDbCourseLog.

6.2.2 *SkyServer Query Log*

The original SkyServer query log for 2016 consists of 12.9 million queries. The clustering procedure considers only queries which have both an AAB and an FB query representation. Having the same input data for a comparison of different query-representation schemes is a prerequisite for meaningful results. As mentioned in Section 4, the current version of AAB does not process queries with arithmetic operations in the WHERE clause, and we also exclude queries without a predicate in the filtering condition. Table 3 summarizes the queries included in the comparison and contains explanations for queries which we have not processed.

With query clustering, one wants to obtain meaningful results, i.e., finding user interests in our case. So we also exclude queries with the following characteristics from further consideration:

- *Queries issued by robots performing a sliding window search (SWS).* To identify this behavior, we have performed a procedure similar to the one described in [6]. The Appendix contains a description. As defined earlier, an SWS is a sequence of queries of identical structure, performing a range search. Here, identical structure means that only parameter values are different, and the ranges are contiguous.

How SWS queries – if included –affect clustering, depends on the similarity function used, AABovl or AABcl. In case of AABovl, SWS queries do not form a cluster, because SWS imply disjoint filtering conditions (no overlap). Hence, for AABovl it counts as noise – more queries are processed, which increase the runtime. With AABcl, SWS queries could form a cluster, because “neighbour” queries will get non-zero similarities. In our opinion, this is not a result one needs to get since SWS represent the information need only of one user, not the common interest of many people.

The procedure of excluding SWS requires a threshold value as parameter which specifies the strictness when looking for SWS. Here, we fix the value to 100, i.e., 100 contiguous range queries from one user in a row are an SWS. This kind of queries occupies 62% of the SkyServer query log. See Table 3. We for our part leave aside such queries since they represent interests of very few users or of even only one, and it even is unclear what the true interest behind an SWS actually is.

- *Queries issued by many users when they open the SkyServer web interface for the first time.* To illustrate,

```
SELECT … FROM fGetNearbyObjEq(258.25,64.05,3) n,
PhotoPrimary p WHERE n.objID=p.objID
```

has been issued 647907 times. It is available at the radial search web page of SkyServer[7], with exactly these default values. Thus, the full log after cleaning, named FullLog, consists of 1.37 million queries.

Obtaining WB query representations even for a log of 1.3 million queries is difficult to impossible. As mentioned, one would need to evaluate all queries to this end. The overall runtime for all queries from the full cleaned log, the *FullLog*, are around 220 days according to SkyServer metadata, the sum of the numbers of rows in all results is about 7.7 billion. Thus, for the WB query representation we have sampled *FullLog*, obtaining *SampledLog*. For this sampling, we have chosen one tenth of the cleaned log. Table 3 of the Appendix is a description of the WB sampled dataset. From now on, we will use the names for the different query logs as in the Appendix, Table 4.

6.3 Evaluation Techniques

According to [21], validity measures for clustering fall into two groups:

- External measures are used when a ground truth is available. The idea behind these measures is to compare the clustering result with the ground truth. An example of such a measure is the Jaccard index [22]. The set of all pairs of objects from two clustering results C and C' is the

TABLE 3

Description of the SkyServer Log data

Property	Value	Share
Size of original query log	12,917,940	
Preprocessed queries (FB, AAB), SkSLog	**10,289,990**	**79.7 %**
Non-preprocessed queries (FB, AAB)	2,627,950	20.3 %
Among non-processed		
Arithmetic operation in WHERE clause	1,158,375	9.0 %
JSQL Parser limitations	784,798	6.1 %
Queries to meta-tables	364,505	2.8 %
Queries without WHERE clause	344,552	2.7 %
Errors in SkyServer logging	17,956	0.1 %
Queries to non-existing tables	2,444	0.02 %
Cleaned queries, FullLog	**1,368,232**	**10.6 %**
Queries excluded	8,921,758	69.1 %
Among excluded		
Requests made by robots performing SWS	8,001,943	62 %
Requests which refer to SkyServer web pages with default values	919,815	7.1 %

[6] https:/ / www.ipd.kit.edu/ arzamaso/ qlc/ readme.html

[7] skyserver.sdss.org/ dr12/ en/ tools/ search/ radial.aspx

disjoint union of the following sets:
S_{11} = {pairs that are in the same cluster under C and C'}
S_{00} = {pairs that are in different clusters under C and C'}
S_{10} = {pairs that are in the same cluster under C but in different ones under C'}
S_{01} = {pairs that are in different clusters under C but in the same under C'}

The values n_{11}, n_{00}, n_{10} and n_{01} are the cardinalities of these sets. The Jaccard index now quantifies the similarity of two clustering results as follows:

$$J(C, C') = \frac{n_{11}}{n_{11} + n_{10} + n_{01}} \tag{19}$$

The index takes values from 0 to 1. The bigger it is, the higher is the similarity.

- Internal measures do not require a ground truth. They rely on criteria derived from the data itself, e.g., intracluster and intercluster distances. In our internal evaluation, we use the BetaCV measure [23], the ratio of the mean intercluster distance over the mean intracluster distance. A small value indicates higher clustering quality. To validate the consistency within clusters, we have used the Silhouette coefficient $s(i)$ [24], which indicates how well each object i lies within its cluster. $s(i)$ takes values from -1 to 1. The bigger $s(i)$, the better i matches its cluster.

6.4 Implementation

To get the query representations, we first parse the queries. We use the JSQL Parser[8] written in Java. Query representations are then stored in the database. All similarity functions are implemented in SQL. To evaluate the results (i.e., to calculate the BetaCV coefficient etc.), we have again used Java.

6.5 Experiments with Supervision

Regarding the student exercise, since the students were asked to perform 4 tasks, we expect to get 4 clusters in the results. Hence, it may make sense to use the k-medoids clustering algorithm with $k = 4$ for each similarity function and corresponding query representation. Table 4 contains the results. As similarity measures, the Jaccard index and the BetaCV coefficient have been used.

The Jaccard indexes for the WB and the FB query representation, which indicate the closeness of the clusters to the ground truth, do not show good results relative to the other approaches. We see three reasons for this:

The database schema consists of only three tables. The tasks have been constructed to have more than one table in an answer SQL statement. There also are only a few attributes in each relation, namely 3, 7 and 3. Hence, the probability of having the same tables and filtering attributes for different tasks is high. This causes a problem with the FB approach. The value of 0.454 of BetaCV for the FB query representation indicates this.

The database has been very small as well – it has 28 rows from 3 tables. This leads to a high probability that queries from different tasks share the same tuples. This in turn affects WB clustering.

The students have made mistakes when formulating queries. If all answers were precise, we would have obtained zero BetaCV for each query representation: While the text of two correct answers might differ, all query representations for one task would be identical. For instance, when looking at the WB representation, a correct query should return only certain tuples. The same holds for FB and AAB. This would have lead to zero intercluster distances. However, the actual average intercluster distances are above zero. See Table 4.

Since the AAB query representation does not rely very much either on metadata (database schema) or on actual data (witnesses), the respective clustering corresponds to the ground truth very well. We have obtained identical results for both AABovl and AABcl because of the high specificity of the tasks: Everybody has been asked to do the same, and mistakes by formulating wrong filtering conditions are unlikely. However, the AAB query representation (as well as FB and WB) cannot cope with the third problem (errors in the student answers). This is in line with our expectations.

To sum up, this experiment shows that all query representations lead to meaningful clustering in theory. However, there are certain obstacles which have turned out to be spoilers: These are the small database schema for FB and the very small database for WB. On the other hand, the results are in line with the limitations we have already discussed when introducing the query representations.

6.6 Experiments with SkyServer

With SkyServer, we have generated the three query representations described before for the randomly sampled log, dubbed *SampledLog*. We first discuss the results generated from these. We also have results with *FullLog*, for the FB and AAB representations, and we describe them as well. Next, by comparing the results from the sampled log to the ones from the regular log, we evaluate how sampling affects the clustering results. Finally, we conduct a study with a domain expert to interpret our results. All these experiments yield different insights regarding the usefulness of the query representations and the appropriateness of the various clustering algorithms when applied to a real-world SQL query log.

6.6.1 *Clustering Results*

Table 5 of the Appendix lists the parameter values for DBCSAN, k-medoids and CLINK. To set them, we rely on the expectation that the size of clusters should be in line with the size of input data. Thus we have set the value of parameter *minPts* of DBSCAN to 0.05% of the number of objects in a dataset. The size of a dataset is the number of distinct objects which have at least one non-zero similarity value in the corresponding proximity matrix. This is because only these objects have a chance to be contained in a cluster. If the data is noisy, and there are many objects without similar ones, no group of similar objects is big enough to form a cluster. For the experiments with full data, FullLog, we set *minPts*=100. This is because we want to compare both the AABcl and the AABovl approach with the same non-strict parameters, i.e. when $minPts = 100 <$ 0.05% of the number of objects in both

[8] http://jsqlparser.sourceforge.net/

cases. We set *eps*= 0.7 for DBSCAN, since it captures sufficient overlap for the AABovl and the WB approach and allows to catch queries close to each other with AABcl. The number of clusters DBSCAN returns will be the value of our parameter k when running k-medoids. While the actual number of clusters is typically not available in most real settings, we use it here nevertheless. This is because we are interested in how good the results can actually be. For the hierarchical CLINK algorithm, the cut-off threshold is equal to *eps* of DBSCAN. For the FB approach however, we follow another strategy. Since FB yields patterns of user behavior, it does not make sense to mix several patterns. Therefore we have chosen the parameters so that only very similar patterns (eps = 0.1) go to the same cluster. Our clustering results consist of almost 1000 clusters: 508 for AABovlFull, 82 for AABovlSampled, 182 for AABclFull, 88 for AABclSampled and 125 for WB.

Table 5 lists the values of the BetaCV coefficient. They are relatively large. This is because we have considered only large clusters for their calculation; the size of a cluster must be no less than 0.05% of the size of a query log. Within such big clusters, a lot of queries have zero similarity with each other. This means that a Query q has an overlap with only a few queries $\{q'_1 \dots q'_n\}$. They are similar to other ones $\{q''_1 \dots q''_m\}$, though q is not similar to them. Put differently, the clusters are not dense. Indeed, they cannot be since the corresponding proximity matrices are sparse. See Fig. 2 of the Appendix with the similarity distributions. Table 6 of the Appendix reports on average Silhouette coefficients.

Nevertheless, for AABovl and WB, clustering yields areas of interests which are small compared to the whole data space. Tables 6 and 7 list aggregated query representations of the biggest clusters (in terms of number of queries) of DBSCAN, with *SampledLog* and *FullLog*. The representation is the minimum bounding rectangle (MBR) of all queries in the cluster. To present the results of WB clustering, one cannot utilize the corresponding query representations – they contain huge numbers of tuples. Instead, we use the more compact and intuitively understandable AAB representation as well. For AAB there also is an area-coverage value available. It is the ratio of the volume of the aggregated access area over the volume of the tables the queries from the cluster are applicable to:

$$areaCoverage = \frac{V_{access}}{V_{content}} \tag{20}$$

To obtain V_{access}, we take bounds of each attribute occurring in a filtering condition of the cluster. So $V_{content}$ is calculated taking the domains of each such attribute.

6.6.2 *Discussion of Query Representations and Clustering Algorithms*

We discuss the usefulness of the various query representations when clustering a real-world query log based on the experiments with SkyServer.

WB clustering. We observe that WB clusters are precise. This is because all queries in the WB clusters ask for the same attributes, the spatial attributes *dec* and *ra*. This means that there have not been any "accidental" similarities, i.e., queries which refer to different attributes sharing witnesses by chance. With these identical attributes, queries whose filtering conditions overlap are similar.

ABovl clustering. We find it remarkable that the aggregated access areas for AABovl and WB similarity are very much alike. Three of the four biggest clusters with these approaches point to the same parts of the sky. We conclude that the AAB query representation and AABovl similarity function also are valid and precise. With AAB being scalable, we for our part conclude that it may be preferable to WB.

A difference we have observed in the AABovl and WB clustering results (with *SampledLog*) is that there are clusters in AABovl which do not exist in WB. The queries inside these clusters have empty results. Of course, WB cannot detect them. For example, the fifth biggest cluster of AABovl has the following aggregated access area:

```
photoprimary.dec ≥ -7.073 ∧ photoprimary.dec ≤ -7.026 ∧
photoprimary.ra ≥ 78.1498 ∧ photoprimary.ra ≤ 78.195
```

Indeed, there is no data object in this area. However, this has not prevented a significant number of AAB query representations from forming a cluster.

One might wonder why clusters with an overlap (like 2 and 3 of AABovl *FullLog*, see Fig. 5) are not one single cluster. We had a closer look at this phenomenon and have observed that queries from one cluster (2) indeed are similar to queries from the other cluster (3). However, these are points that are density-reachable, not core points

TABLE 4
The results of the experiments with ground truth, dataset GtDbCourseLog, clustering algorithm k-medoids, k = 4

	WB	FB	AABovl	AABcl
Jaccard index (compared with ground truth)	0.7518	0.4339	**0.9451**	**0.9451**
Experiment				
BetaCV	0.257	0.454	**0.1402**	**0.1402**
Average intercluster distance	0.194	0.317	0.1402	0.1402
Average intracluster distance	0.753	0.698	1	1
Average Silhoette coefficient	**0.885**	0.409	0.87	0.85
Ground truth				
BetaCV	**0.171**	0.194	0.231	0.231
Average intercluster distance	0.132	0.167	0.231	0.231
Average intracluster distance	0.773	0.857	1	1
Average Silhoette coefficient	0.74	**0.754**	0.689	0.689

TABLE 5
Values of BetaCV coefficient

Dataset	Algorithm	WB	AABovl	AABcl
SampledLog	DBSCAN	0.933	**0.925**	0.993
	K-medoids	0.9995	0.9997	0.9998
	CLINK	0.998	0.997	0.999
FullLog	DBSCAN	-	**0.913**	0.981

in DBSCAN terminology. To conclude, density reachability is a characteristic that is not sufficient to end up in the same cluster in general.

AABcl clustering. We have obtained big clusters which cover significant parts of the data space when clustering with the AABcl similarity function. This is plausible: As Fig. 5 shows, the third biggest cluster of AABcl in the sampled log (the one with Rank 3) has a big rectangular part from to 41.269 to 84.973 in the *dec* column. This part is due to several queries with broad diapasons: These queries request data based on attributes *ra* and *dec* with broad ranges. Different users have issued these queries, so they are not SWS. They act like "supermassive" objects and have a "gravity effect" on queries with smaller ranges in the neighborhood. In contrast to AABovl, where supermassive objects do not have sufficient overlap with small objects to fall into a cluster, AABcl is sensitive to queries with broad ranges. It is also sensitive to sliding window search. However, because we had filtered them out beforehand, we did not observe the influence of SWS on AABcl clustering. Summing up, whether AABcl is successful strongly depends on specifics the query log: It needs to be free from massive downloading, i.e., sliding window search (SWS), and there should not be any very broad range queries. Put differently, this also indicates that cleaning the query log before analysis might yield better, more meaningful results.

FB clustering. Clustering in line with the FB paradigm reveals patterns of SkyServer database usage, i.e., which tables, views, UDFs and filtering attributes individuals tend to use. However, as mentioned before, this query representation does not reveal areas of the data space users are interested in. This also is why the column "Area coverage" is empty for FB clustering.

Clustering algorithms. Different clustering algorithms have performed differently on the SkyServer log as well. The data for the AAB and WB approaches contains noise – queries which do not have sufficiently many similar objects. Different algorithms have different sensitivity to this kind of noise. DBSCAN is able to work with this noisy data [19]. k-medoids suffers from it a lot since it partitions the data, and all objects end up in some cluster. CLINK is sensitive to noise as well, but ignoring small clusters can solve the problem here: If the data to be clustered contains a lot of outliers, many small or even singleton clusters occur. The algorithm does not merge them to bigger clusters since they are too distant from each other. So we have classified clusters with a size less than a specific value as noise. This is why the clustering result with CLINK does look structurally similar to the one with DBSCAN, containing an extra "cluster" for noise.

Summing up, we would give preference to a density-based clustering algorithm when it comes to query logs, for the following reasons:

1. Data might be noisy.
2. One cannot predict the number of clusters in advance, as required by k-means-based algorithms.

Consequently we have clustered the big log file, *FullLog*, only with DBSCAN.

6.6.3 *Influence of Random Sampling on the Clustering Results*

Clustering large query logs with the procedure used in this article is time-consuming since one has to (1) extract the query representations, (2) build a proximity matrix, and (3) perform the actual clustering. Hence, it might be a good idea to cluster a sample of the data. However, so far it is not clear whether and how sampling influences the result. In particular, it is unclear (1) how the aggregated access areas of clusters using full and sampled data differ, and (2) which clustering results a domain expert finds better. The first answer will be given right away, while the second question is discussed in Section 6.6.4.

AABovl clustering. As Fig. 5 shows, clustering on a sample of the data and on the full data yields similar results with AABovl. The differences mainly have to do with cluster ranks, i.e., the position when sorting clusters by their numbers of objects. This is why not all clusters occur with both the sampled and the full log: They exist, but not in the top 10.

AABcl clustering. The results with AABcl differ more. Fig. 5 shows how certain queries "move" from one cluster to another one. It is safe to say that the closeness approach is less robust when it comes to sampling than the overlap approach. Again, the query which has formed a long vertical rectangle and has gone to the second biggest cluster in the sampled data has not disappeared; it just has gone to a less popular cluster not in the top 10.

FB clustering. As Tables 6 and 7 indicate, sampling does not change the order of the most popular patterns with FB clustering. We have checked the first 50 popular patterns with sampled and full data and have found only one difference. The ranks change only slightly, by 2 positions at most, and they usually remain the same.

Overall, sampling is useful when clustering an SQL query log. If a query log is huge and requires a lot of time to process, sampling can give way to quick insights. However, AABcl is less robust in this respect.

6.6.4 *Feedback from a Domain Expert: Clustering Interpretation*

As mentioned, a good clustering must be interpretable. Here, this means that each cluster should relate to a particular user interest. In astronomy, this means that a cluster may contain several astronomical objects, but they all must form a single astronomic category, like "North galactic pole" or "Lockman hole", i.e., represent one research trend. To investigate how successful our clustering has been, and whether it reflects user interests, we have asked a domain expert to inspect our results. He is an astronomer from the Max Planck Institute for Astronomy in Heidelberg, Germany. At the same time, to ease the process of cluster interpretation and ensure a complete representation of the interests of the astronomical community, we have made use of another important astronomical data source, the Simbad astronomical database[9]. We use it as a reference point. Simbad provides information on astronomical objects which have been studied in scientific publi-

[9] http://simbad.u-strasbg.fr/simbad/

TABLE 6
Top clusters of DBSCAN, dataset SampledLog

#	Relative size	Area coverage	Aggregated query representation
AABovl			
1	0.78%	0.25%	photoprimary.dec ≥ 1.2 ∧ photoprimary.dec ≤ 7.3 ∧ photoprimary.ra ≥ 10.3 ∧ photoprimary.ra ≤ 18.8
2	0.08%	0.02%	photoprimary.dec ≥ 54.8 ∧ photoprimary.dec ≤ 56.8 ∧ photoprimary.ra ≥ 241.4 ∧ photoprimary.ra ≤ 245
3	0.07%	0.002%	photoprimary.dec ≥ -9.1 ∧ photoprimary.dec ≤ -9.05 ∧ photoprimary.ra ≥ 120 ∧ photoprimary.ra ≤ 120.05
4	0.06%	$3.7 \bullet 10^{-8}$%	photoprimary.dec ≥ 14.839 ∧ photoprimary.dec ≤ 14.84 ∧ photoprimary.ra ≥ 2.023∧ photoprimary.ra ≤ 2.024
AABcl			
1	7.97%	81%	photoobj.dec ≥ -42.147 ∧ photoobj.dec ≤ 76.686 ∧ photoobj.ra ≥ 0 ∧ photoobj.ra ≤ 359.821
2	3.53%	18.4%	photoprimary.dec ≥ -2.7 ∧ photoprimary.dec ≤ 59.6 ∧ photoprimary.ra ≥ 0 ∧ photoprimary.ra ≤ 73
3	3.06%	54.3%	photoprimary.dec ≥-4.94 ∧ photoprimary.dec ≤ 91 ∧ photoprimary.ra ≥ 0 ∧ photoprimary.ra ≤ 360
4	2.97%	93.35%	photoobjall.dec ≥-60.572 ∧ photoobjall.dec ≤ 84.98 ∧ photoobjall.ra ≥ 0 ∧ photoobjall.ra ≤ 360
WB			
1	0.6%	0.1%	photoprimary.dec ≥ 1 ∧ photoprimary.dec ≤ 7.8 ∧ photoprimary.ra ≥ 9.6 ∧ photoprimary.ra ≤ 19.4
2	0.08%	0.03%	photoprimary.dec ≥ -1.5 ∧ photoprimary.dec ≤ 1.2 ∧ photoprimary.ra ≥ 350.8 ∧ photoprimary.ra ≤ 353.1
3	0.06%	0.01%	photoprimary.dec ≥ 54.4 ∧ photoprimary.dec ≤ 56.7 ∧ photoprimary.ra ≥ 240.9 ∧ photoprimary.ra ≤ 245
4	0.05%	$5.1 \bullet 10^{-6}$%	photoprimary.dec ≥ -9.105 ∧ photoprimary.dec ≤ -9.057 ∧ photoprimary.ra ≥ 120.009 ∧ photoprimary.ra ≤ 120.054
FB			
1	27.77%		{specobj;specobj.bestobjid}
2	17.15%		{photoz; galspecline; specobj; photoz.objid}
3	10.13%		{photoobj; photoobj.dec; photoobj.ra}
4	10.11%		{phototag; fgetobjfromrecteq; phototag.objid}

TABLE 7
Top clusters of DBSCAN, Dataset FullLog

#	Relative size	Area coverage	Aggregated query representation
AABovl			
1	1.37%	0.19%	photoprimary.dec ≥ 9.3 ∧ photoprimary.dec ≤ 16.8 ∧ photoprimary.ra ≥ 17.8 ∧ photoprimary.ra ≤ 29
2	0.89%	0.18%	photoprimary.dec ≥ 0.6 ∧ photoprimary.dec ≤ 8.2 ∧ photoprimary.ra ≥ 9.6 ∧ photoprimary.ra ≤ 19.8
3	0.38%	0.03%	photoprimary.dec ≥-6.2 ∧ photoprimary.dec ≤ -3.2 ∧ photoprimary.ra ≥ 32.8 ∧ photoprimary.ra ≤ 38.1
4	0.27%	0.1%	photoobjall.dec ≥ 1.8 ∧ photoobjall.dec ≤ 8.4 ∧ photoobjall.ra ≥ 4 ∧ photoobjall.ra ≤ 10.8
AABcl			
1	4.35%	11.27%	photoprimary.dec ≥ -11.5 ∧ photoprimary.dec ≤ 59.6 ∧ photoprimary.ra ≥ 0 ∧ photoprimary.ra ≤ 76.4
2	3.06%	93.7%	photoobjall.dec ≥ -61.551 ∧ photoobjall.dec ≤ 84.98 ∧ photoobjall.ra ≥ 0 ∧ photoobjall.ra ≤ 360
3	1.26%	15.76%	photoprimary.dec ≥-24.323 ∧ photoprimary.dec ≤ 84.973 ∧ photoprimary.ra ≥ 279.4 ∧ photoprimary.ra ≤ 360
4	1.16%	99.72%	apogeestar.dec ≥ -90 ∧ apogeestar.dec ≤ 87.581 ∧ apogeestar.ra ≥ 0.833 ∧ apogeestar.ra ≤ 360
FB			
1	27.85%		{ specobj.bestobjid; specobj }
2	17%		{ photoz; galspecline; specobj; photoz.objid }
3	10.12%		{ phototag; fgetobjfromrecteq; phototag.objid }
4	10.07%		{ photoobj; photoobj.dec; photoobj.ra }

cations in astronomy. It has 12 tables and contains 9.3 million astronomical objects outside of our solar system and 340 thousand bibliographic references. There are some characteristics common for each astronomical object:

- Basic data: object types, coordinates and other astronomical features; General bibliography for the object, including references to all published papers from the journals scanned regularly, currently about 80 titles.

Naturally, two astronomical databases, SkyServer and Simbad, are expected to have a big overlap of the objects they contain. This also holds for attributes like special coordinates, object types etc. However, they are constructed very differently and partly based on different data, so they are quite independent at the same time. With the help of the domain expert, we have mapped our clustering result to the Simbad database. Almost every cluster from our results filters spatial coordinates right ascension *ra* and declination *dec*. We have plotted the clusters on the *ra-dec* plane and have mapped them to the *ra-dec* density map of astronomical publications.[10]

Of course, one cannot expect a perfect overlap. Not every astronomer looks at the data from SkyServer when writing an article. And vice versa – some data from SkyServer may have been queried for by laymen or high school students, without the publication of a paper. However, both our clusters and Simbad data should reflect hot spots in astronomy. Thus, a relatively high correlation is

[10] A related idea would be to not only look at the actual Simbad data, but also at its query log. However, we have not been able to obtain this data because of privacy concerns of scientists responsible for Simbad.

better as an experiment result, according to our perception. As we have pointed out earlier, our clustering results consist of around 1000 clusters. Identifying user interest in each of them is a daunting task for any domain expert. Such an identification takes our expert 10 minutes on average per cluster, mainly depending on the number of astronomical objects in the cluster. To make manual inspection feasible, we have selected the top 15 clusters from each approach (*AABovlFull*, *AABovlSampled*, *AABclFull*, *AABclSampled* and *WB*) which have the most overlap with Simbad data, i.e., clusters which 'repeat' the high density areas of Simbad. We assume that these clusters are the most interesting ones for domain experts: There is a high number of publications on the astronomical objects from these particular parts of the sky. Fig. 6 graphs them together with Simbad data. We have mapped the queries in the various clusters to Simbad data of published papers in the ra-dec plane. A query in the figure is a rectangle which includes admissible ra-dec values. Note that the number near a cluster indicates its' rank: the biggest cluster (in terms of number of queries) has Rank 1. The sixth figure is the pure density map of publications (Simbad). Dark grey areas stand for a high amount of publications; for light grey, the picture is different. One can see that queries from the clusters indeed repeat the distribution of Simbad data: The clusters are located in the grey areas of the Simbad map.

For each cluster, we plot the overlap of the individual query areas on the map with all the Simbad entries. This allows the domain expert to estimate whether the cluster contains one or several astronomic categories of well-studied objects. Having inspected the clusters obtained with DBSCAN, our domain expert has concluded that they are quite different. From his point of view, there are:

- *Large clusters*[11], each of which covers more than 3% of the sky or several hundreds or thousands of square degrees.[12] According to our expert, none of them can be associated with a specific scientific goal or type, i.e., there is no corresponding single user interest. In what follows, we refer to these clusters as *LwoUI* clusters (Large clusters WithOut User Interest). Other large clusters consist of several very small areas each of which contains a single Simbad entry. For these clusters, our expert has identified a specific user interest. We call these clusters *LwUI* clusters (Large clusters With User Interest).
- *Intermediate clusters*, each of which covers less than 3% of the sky. As before, we call them *IwUI* (Intermediate clusters With User Interest) if they contain user interests and *IwoUI* (Intermediate clusters WithOut User Interest) otherwise. The domain expert has observed that these clusters have a size so that they likely correspond to a specific scientific goal or type.
- *Extremely small clusters*, which typically consist of several queries referring to the same individual object and cover around 0.01% of the sky. We consequently dub these clusters *ESwoUI* and *ESwUI*.

Table 7 in the Appendix reveals how many clusters of each category the five approaches (*AABovlFull*, *AABovlSampled*, *AABclFull*, *AABclSampled* and *WB*) identify. The table also lists the astronomical objects from the various clusters.

For each scheme, the domain expert has ranked each cluster among the 15 most populated ones according to the probability that it properly covers a region of the sky of particular interest. We have then averaged the grades to rank the five schemes. They take values from one to ten, with ten being the highest interest. *AABovlFull* has the highest average score, followed by *AABclFull*, *AABovlSampled* and *AABclSampled*. *WB* is last, see Table 7 in the Appendix.

One can observe that sampling worsens the clustering results for AABovl. Some clusters disappear, not having enough objects as neighbors. Decreasing minPts value will not always help; the following example shows this:

Example 15. Consider the three queries q_1, q_2 and q_3 with $D(q_1, q_2) = 0.5$, $D(q_1, q_2) = 0.8$, and $D(q_1, q_3) = 1$. $eps = 0.9$, $minPts = 2$. All three queries end up in the same cluster. We now sample the log and exclude q_2. Setting $minPts$ to 1 does not yield a cluster of queries q_1 and q_3 because they do not overlap. Setting eps to 1 does not make any sense because then all queries go to the same cluster.

Based on this, we hypothesize that WB, which also is overlap-based, would have given better results if it had taken place on the full data. We conclude that an analytical calculation of overlap, i.e., AABovl, is useful. It provides sufficient accuracy and is scalable. In consequence, one does not have to do sampling, which bogs down the clustering results.

On the other hand, sampling has helped to obtain better results with the closeness approach, AABcl: Clusters have become smaller and, hence, more focused. Thus, sampling allows to better identify user interests than clustering on the original data.

7 CONCLUSIONS

Knowing user interests in a data space is important for database owners and for domain experts. Clustering the query log can yield interesting insights to this end. In this paper we have studied the clustering of SQL query logs. In particular, we have established the design space, i.e., which query representations, which algorithms, which distance measures. Next, we have looked at possible instantiations from the literature systematically and have discussed our expectations for each alternative. We also have proposed new alternatives as well, since the existing proposals have not been fully satisfying. Our new approaches, which we had proposed to do away with the weaknesses of existing approaches, have turned out to be better in most respects. Finally, we have carried out several studies, one with a domain expert in order to arrive at a

[11] Though there is a notion of a galaxy cluster in the domain of astronomy, here and in what follows we always mean a cluster of SQL queries when using the word 'cluster'. An exception is the Column "Extended objects" of Table 7 in the Appendix.

[12] The whole celestial sphere covers 41253 square degrees. Analogously to one degree being equal to $\pi/180$ radians, a square degree is equal to $(\pi/180)^2$, or about $1/3283 = 3.0462 \times 10^{-4}$ steradians (0.30462 msr). To calculate the area of a query cluster in square degrees, one needs to apply the $cos(deg)$ factor, i.e., $S = ra \times cos(deg)$.

ground truth, a feature which we have not observed in any previous work on analyses of database-query logs. The study with the domain expert in particular has revealed the usefulness of clustering when user interests need to be identified.

Our new approach captures query similarity on the data level. Unlike other approaches, "witness based" in particular, it scales relatively well with the size of the log.

Our future work will focus on SQL query recommendation. We plan to leverage our new insights regarding query similarity to find similar user sessions from which query suggestions can be generated. While SQL query recommendation has already been investigated earlier [25], [26], [27], revisiting the topic based on this current study might reveal new insights.

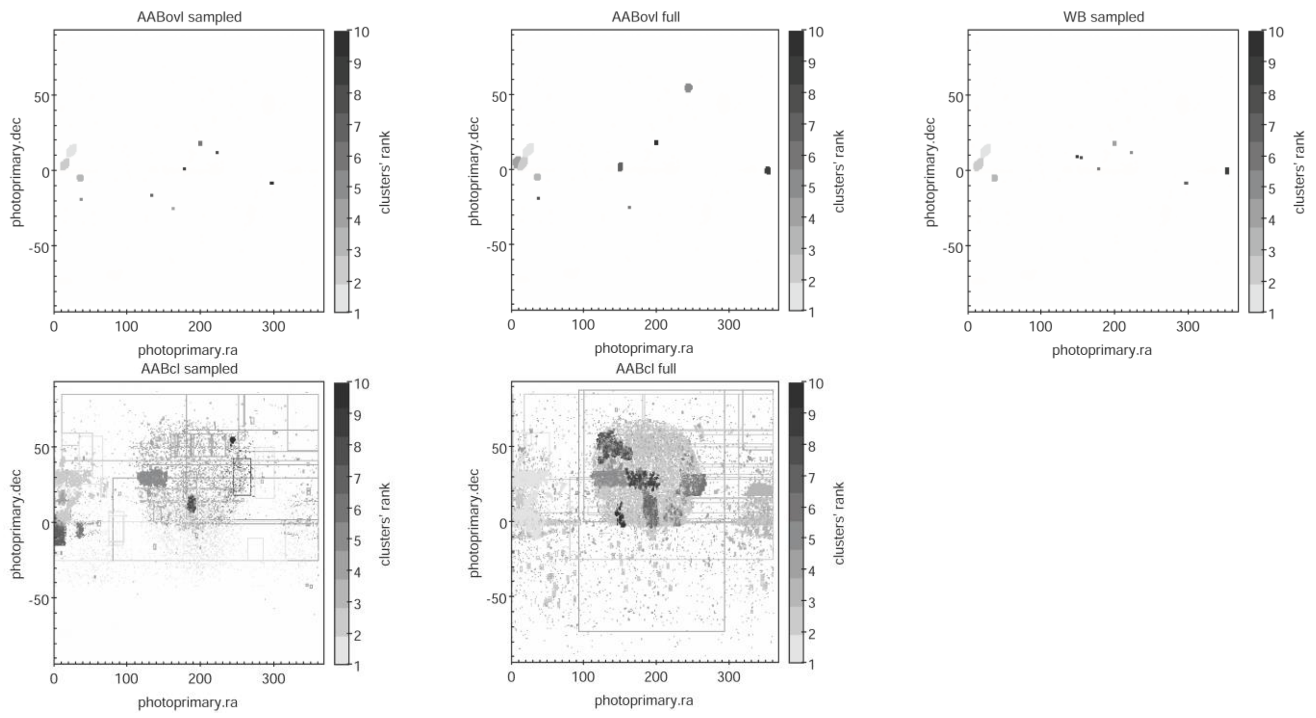

Fig. 5. Clustering results, DBSCAN algorithm

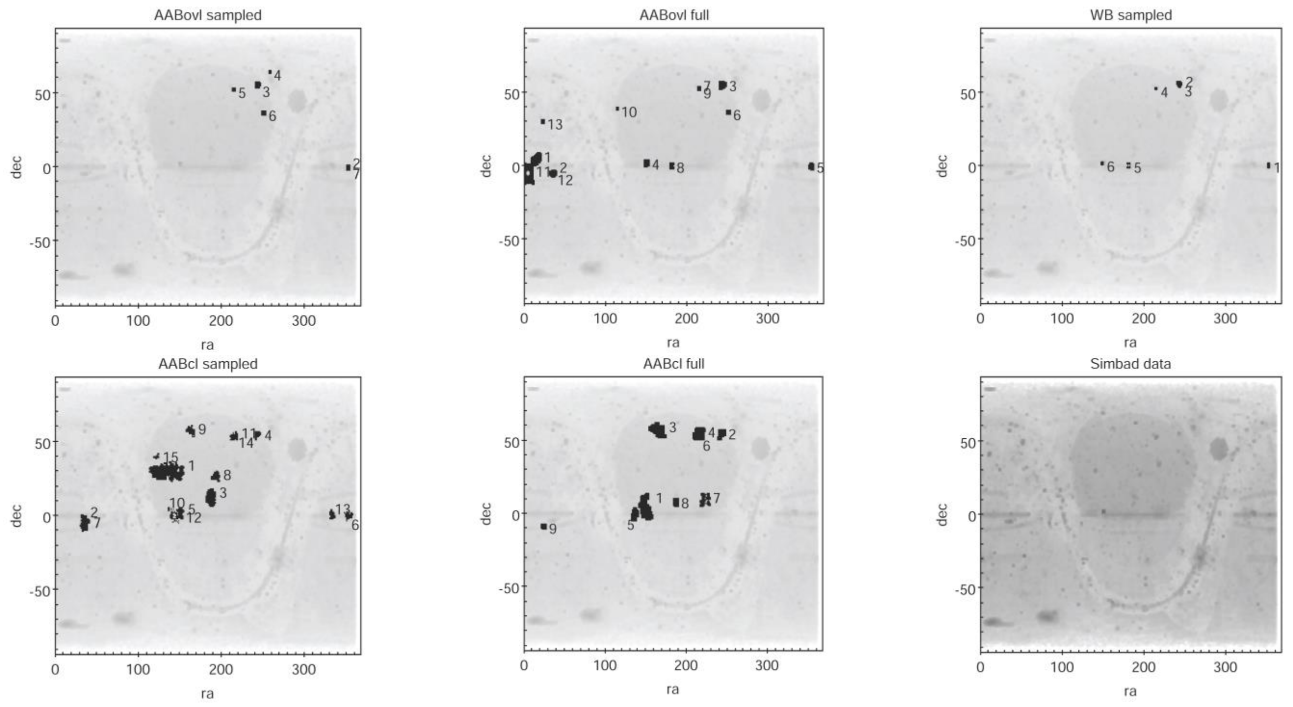

Fig. 6. Mapping clustering results to Simbad data

REFERENCES

[1] N. Khoussainova and Y. Kwon, "Snipsuggest: Context-aware autocompletion for sql," *Proc. VLDB Endow.*, vol. 4, no. 1, pp. 22–33, 2010.

[2] J. Akbarnejad *et al.*, "SQL QueRIE recommendations," *Proc. VLDB Endow.*, vol. 3, no. 1–2, pp. 1597–1600, 2010.

[3] H. V. Nguyen, B. Goldman, K. Böhm, G. Hinkel, F. Becker, and E. Müller, "Identifying User Interests within the Data Space – a Case Study with SkyServer," *Int. Conf. Extending Database Technol.*, pp. 641–652, 2015.

[4] M. Halkidi, "On Clustering Validation Techniques," *J. Intell. Inf. Syst.*, vol. 173, no. 2, pp. 107–145, 2001.

[5] R. Xu, "Survey of clustering algorithms for MANET," *IEEE Trans. Neural Networks*, vol. 16, no. 3, pp. 645–678, 2005.

[6] N. Arzamasova, M. Schäler, and K. Böhm, "Cleaning Antipatterns in an SQL Query Log," *IEEE Trans. Knowl. Data Eng.*, vol. XX, no. XX, pp. 1–14, 2017.

[7] X. Wang, A. Meliou, and E. Wu, "QFix: Diagnosing errors through query histories," in *Proc. 2017 ACM SIGMOD Int. Conf. Manag. data - SIGMOD '17*, 2017, pp. 1369–1384.

[8] J. Aligon, M. Golfarelli, P. Marcel, S. Rizzi, and E. Turricchia, "Similarity measures for OLAP sessions," *Knowl. Inf. Syst.*, vol. 39, no. 2, pp. 463–489, 2014.

[9] X. Yang, C. M. Procopiuc, and D. Srivastava, "Recommending join queries via query log analysis," *Proc. - Int. Conf. Data Eng.*, pp. 964–975, 2009.

[10] Z. Chen and T. Li, "Addressing diverse user preferences in SQL-query-result navigation," *Proc. 2007 ACM SIGMOD Int. Conf. Manag. data - SIGMOD '07*, p. 641, 2007.

[11] A. Islam and D. Inkpen, "Semantic text similarity using corpus-based word similarity and string similarity," *ACM Trans. Knowl. Discov. Data*, vol. 2, no. 2, pp. 1–25, 2008.

[12] A. Kamra, E. Terzi, and E. Bertino, "Detecting anomalous access patterns in relational databases," *VLDB J.*, vol. 17, no. 5, pp. 1063–1077, 2008.

[13] V. H. Makiyama, M. J. Raddick, and R. D. C. Santos, "Text mining applied to SQL queries: A case study for the SDSS SkyServer," *CEUR Workshop Proc.*, vol. 1478, pp. 66–72, 2015.

[14] G. Kul, D. Luong, T. Xie, P. Coonan, V. Chandola, and ..., "Ettu: Analyzing query intents in corporate databases," *Proc. 25th ...*, 2016.

[15] G. Kul, G. S. Member, D. Thanh, A. Luong, T. Xie, and V. Chandola, "Similarity Metrics for SQL Query Clustering."

[16] C. Science, "A Data-Centric Approach to Insider Attack," pp. 382–401, 2010.

[17] Defays, "An efficient algorithm for a complete link method," *Comput. J.*, no. January, pp. 364–366, 1977.

[18] H. S. Park and C. H. Jun, "A simple and fast algorithm for K-medoids clustering," *Expert Syst. Appl.*, vol. 36, no. 2 PART 2, pp. 3336–3341, 2009.

[19] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A Density-based Algorithm for Discovering Clusters a Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 1996, pp. 226–231.

[20] B. Chandra, B. Chawda, B. Kar, K. V. M. Reddy, S. Shah, and S. Sudarshan, "Data generation for testing and grading SQL queries," *VLDB J.*, vol. 24, no. 6, pp. 731–755, 2015.

[21] M. Halkidi, Y. Batistakis, and M. Vazirgiannis, "Cluster validity methods," *ACM SIGMOD Rec.*, vol. 31, no. 2, p. 40, 2002.

[22] S. Wagner and D. Wagner, "Comparing Clusterings - An Overview," *Analysis*, vol. 4769, no. 1907, pp. 1–19, 2007.

[23] W. M. J. Mohammed J. Zaki, *Data Mining and Analysis: Fundamental concepts and Algorithms*. 2014.

[24] J. G. Pearce, Z. Shaar, and R. E. Crosbie, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *J. Comput. Appl. Math.*, vol. 20, pp. 53–65, 1986.

[25] M. Eirinaki, S. Abraham, N. Polyzotis, and N. Shaikh, "QueRIE: Collaborative database exploration," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 7, pp. 1778–1790, 2014.

[26] J. Aligon, E. Gallinucci, M. Golfarelli, P. Marcel, and S. Rizzi, "A collaborative filtering approach for recommending OLAP sessions," *Decis. Support Syst.*, vol. 69, pp. 20–30, 2015.

[27] M. Eirinaki and S. Patel, "QueRIE reloaded: Using matrix factorization to improve database query recommendations," *Proc. - 2015 IEEE Int. Conf. Big Data, IEEE Big Data 2015*, pp. 1500–1508, 2015.

Natalia Arzamasova has received her diploma degree from Chuvash State University (Russia), worked as a software engineer in Vocord Software Company, creating automatic enterprise databases, services and user interface. Currently she is working in Karlsruhe Institute of Technology (KIT), Germany, on her Ph.D. Her research interests include query log analysis.

Klemens Böhm is full professor at Karlsruhe Institute of Technology (KIT), Germany, since 2004. Prior to that, he has been affiliated with University of Magdeburg, Germany, ETH Zurich, Switzerland, and GMD – Forschungszentrum Informationstechnik GmbH, Darmstadt, Germany. Current research topics at his chair are knowledge discovery and data mining in big data, data privacy and workflow management.

Bertrand Goldman Bertrand Goldman is a staff member of the Max-Planck-Institut for Astronomie in Heidelberg, Germany, since 2004. Prior to that, he has studied at the Centre d'Études Atomiques in the Particle physics department, before joining New Mexico State University and NASA Ames Research Centre as a postdoc researcher. He has studied the properties of low-mass stars and the content of the Solar neighbourhood using large catalogues and data mining technics.

Christian Saaler Christian Saaler recently received his Bachelor's degree from the Karlsruhe Institute of Technology (KIT). His thesis discussed different possible clustering techniques for analyzing query logs. Currently he is working in the field of software engineering with a deep interest in data analytics.

Martin Schäler received his Master degree from Otto-von-Guericke University Magdeburg, Germany, in 2010. Afterwards he was employed as a research assistant and scientific coordinator at the same university, receiving his Ph.D. degree in 2014. Since August 2015 he is a post-doctoral researcher at Karlsruhe Institute of Technology (KIT), Germany. His research interests include multidimensional access methods, hardware-sensitive database tuning and provenance.