Learning Analysis Behavior in SQL Workloads

Clement Moreau, Veronika Peralta
University of Tours
Blois, France
firstname.lastname@univ-tours.fr

ABSTRACT

This paper presents a set of analyses aiming at better understanding the SQLShare workload [13] and learning users' analysis behavior. SQLShare is a database-as-a-service platform targeting scientists and data scientists with minimal database experience, whose workload was made available to the research community. According to the authors of [13], this workload is the only one containing primarily ad-hoc hand-written queries over user-uploaded datasets. In this paper we analyze this workload, by comparing users' explorations (sequences of queries), looking for common SQL operations performed by the users during data analysis. We use a clustering algorithm to retrieve groups of similar explorations and we analyze the obtained clusters through many statistical and visual indicators for explaining analysis patterns inside clusters. To our knowledge, this is the first attempt to characterize human analysis behavior in SQL workloads.

1 INTRODUCTION

The analysis of a database workload to support Interactive Database Exploration (IDE) [12] receives increasing interest as it offers many practical interests, from the monitoring of database physical access structures [5] to the generation of user-tailored collaborative query recommendations for interactive exploration [8, 21].

Characterising user behavior while analysing data, i.e. learning the way users analyse data (the type and order of operations, the level of detail, the degree of focus) is a step forward in the understanding of analysis activities and offers new applications, for instance to understand users' information needs, to identify "struggling" during the exploration, or to provide better query recommendations. Notably, IDE systems usually do not offer such facilities. The prediction of next analysis steps is particularly interesting, enabling beforehand execution of probable queries and caching of results, as well as advanced optimization strategies. Finally, we mention the detection of clandestine intentions [2] as another potential benefit. Indeed, as reported by [2], query sequences may reflect such intentions, where users prefer to obtain information by means of sequences of smaller, less conspicuous queries to avoid direct queries which may disclose their true interests. The identification of typical analysis patterns may help distinguishing normal from clandestine intentions.

In this paper we deal with the identification of analysis patterns in a log of SQL explorations devised by real users. We consider that an exploration is a coherent sequence of queries over a database schema, done by a user with the goal of fulfilling an information need. We experiment on the SQLShare workload of hand-written[1] queries over user-uploaded datasets [13]. In particular, we use the segmentation of the SQLShare workload in coherent explorations proposed in [29].

Some previous works consider analysis patters within OLAP explorations. In [30], Rizzi and Gallinucci described 4 recurrent types of user analyses and propose a tool for generating realistic explorations based on these usage types. In [24], we cluster together explorations showing similar analysis patterns, learning 11 analysis patterns from OLAP workloads devised by students and expert analysts.

The idea behind analysis patterns is to look for sequences of common operations performed together when analysing data, as some kind of movements in a data space. From this point of view, OLAP operations (e.g. drilling down, adding a filter, changing a measure) are first class citizens, while the actual analyzed data is less important. For example, we can retain that a user performed a sequence of drills down, disregarding the dimension that was drilled down or the semantics of the underlying data. Explorations are compared in such terms, i.e. to what extent they share the same sequences of operations and evolve at the same level of aggregation and filtering.

Transposing such approach to regular, non multidimensional SQL workloads raises many challenges. Even if a sequence of SQL queries is issued to explore the database content, non multidimensional relational schemata do not have regularities one expects from the multidimensional model, explorations may not be expressed through roll-up or drill-down operations, SQL queries may deviate from the traditional star-join pattern commonly used for analytical purpose, etc.

In this paper we present an extension of our previous work [24] for learning analysis patterns in SQL workloads. In particular, we reuse and extend similarity measures for comparing queries and explorations, and pair them with clustering algorithms. Contrarily to [24], which uses hierarchical clustering, we combine UMAP and DBSCAN methods, which proved to be well adapted to complex sequences [22]. The obtained clusters are then analyzed using several statistic and visual indicators, allowing to characterize analysis behavior in each cluster.

Our contributions include: (i) a representation of queries and explorations in the space of SQL operations, including similarity functions tailored for SQL queries and explorations (described in Section 3), (ii) a statistical analysis of the SQLShare workload in terms of queries and operations (Section 4), (iii) a proposal for clustering SQL explorations, (Section 5), and (iv) a large analysis of the obtained clusters, via many complementary statistical and visual indicators, revealing several (common and specific) patterns of users' analysis behavior (Section 5).

2 RELATED WORK

In this section we briefly describe the SQLShare workload and we present related work concerning workload analysis and indicators tailored for analyzing sequences. Finally, we discuss clustering algorithms adapted to sequences.

[1] Consistently with the authors of [13], we use the term hand-written to mean, in this context, that the query is introduced manually by a human user, which reflects genuine interactive human activity over a dataset, with consideration between two consecutive queries.

2.1 SQLShare

The SQLShare workload is the result of a Multi-Year SQL-as-a-Service Experiment [13], allowing any user with minimal database experience to upload their datasets on-line and manipulate them via SQL queries. What the authors wanted to prove with this experiment is that SQL is beneficial for data scientists. They observed that most of the time people use scripts to modify or visualize their datasets instead of using the SQL paradigm. Indeed, most user needs may be satisfied by first-order queries, that are much simpler than a script, but have the initial cost of creating a schema, importing the data and so on. SQL-as-a-Service frees the user of all this prior work with a relaxed SQL version.

The SQLShare workload is composed of 11,137 SQL statements, 57 users and 3,336 user's datasets. To the best of our knowledge, as reported by the authors of [13], this workload is the only one containing primarily ad-hoc hand-written queries over user-uploaded datasets. As indicated in the introduction, hand-written means that the query is introduced manually by a human user, which reflects genuine interactive human activity over a dataset, with consideration between two consecutive queries.

The SQLShare workload is analyzed in [13], particularly to verify the following assumption: *"We hypothesized that SQLShare users would write queries that are more complex individually and more diverse as a set, making the corpus more useful for designing new systems."* . The authors showed empirically that the queries in the SQLShare workload are complex and diverse. They also analyzed the churn rate of SQLShare users and conclude that most users exhibit a behavior that suggest an exploratory workload.

Other SQL workloads, as SDSS [32] or REACT-IDA [21] include SQL queries generated with specific GUI or applications. Although generated SQL queries are less richer than hand-written ones [13], the approach presented in this paper can be applied to these workloads, and to smaller ones, as those presented in [17].

2.2 Workload analysis

Other scientific domains close to Database, like Information Retrieval or Web Search, have a long tradition of log analysis aiming at facilitating the searcher's task [37]. Many works extract features from queries or search sessions and use them to disambiguate the session's goal, to generate recommendations, to detect struggling in sessions, etc. Since databases tend to be more used in an exploratory or analysis fashion, as evidenced by the SQLShare workload, it is not a surprise that many recent works pay attention to the analysis of database workloads, in addition to those works analyzing workload for optimization or self-tuning purposes. We present some recent advances in this area, differentiating by the type of logs (OLAP logs and SQL logs).

Analyzing OLAP explorations. Logs of OLAP analyses are simpler than SQL ones in the sense that they feature multidimensional queries that can easily be interpreted in terms of OLAP primitives (roll-up, drill-down, slice-and-dice, etc.). In one of our previous works [31], we proposed an approach for detecting OLAP analyses phrased in SQL, by converting SQL queries into OLAP queries and then checking if two consecutive queries are sufficiently close in terms of OLAP operations. Later, we used supervised learning to identify a set of query features allowing to characterize focus zones in OLAP explorations [7], or to identify queries that better contribute to an exploration [6].

In a more recent work [24], we analyzed OLAP workloads devised by students and expert analysts, looking for sequences of common operations performed together when analysing data. We identified 11 analysis patterns corresponding to different analysis behavior. For example, focused explorations (which regularly increase the level of detail and filtering by adding drill-downs and filters), oscillating explorations (alternating drill-downs and roll-ups with few filters), short explorations with few operations and even explorations with repeated queries. We used a hierarchical clustering algorithm, paired with a Contextual Edit Distance [23] to cluster explorations representing the same behavior.

The present work is a continuation of our previous work, in particular [24]. The main differences are that we make no assumption about the type of queries in the workload (particularly, they may not be multidimensional queries), and we have no ground truth (i.e., no human manual inspection of each query) on the workload.

Analyzing SQL logs. SQL workload analysis has recently attracted attention beyond query optimization, for instance for query recommendation [8], query autocompletion [16], or user interest discovery [26]. All these works use the SDSS workload for their tests. In [21], Milo and Somech identify and generalize relevant previous sessions, in the REACT-IDA workload, to generate personalized next-action suggestions to the user. In [33], they study interestingness measures for mining workloads. Embedded SQL code is analyzed in [34] to measure its quality, mainly for maintainability purpose. The authors quantify quality based on the number of operators (joins, unions), operands (tables, subqueries) and variables in the SQL code, experimenting with SQL codes embedded in PL/SQL, COBOL and Visual Basic.

Jain et al. ran a number of tests on the SQLShare workload [13], some of them being reported above, showing the diversity and complexity of the workload. In [35], Vashistha and Jain analyze the complexity of queries in the SQLShare workload, in terms of the following query features: number of tables, number of columns, query length in characters, numbers of operators (Scan, Join, Filter), number of comparison operators (LE, LIKE, GT, OR, AND, Count), and the query run-time. They define two complexity metrics from these features: the Halstead measure (traditionally used to measure programs complexity) and a linear combination whose weights are learned using regression.

Finally, a recent work investigated various similarity metrics over SQL queries, aiming at clustering queries [17] for better workload understanding. Queries are issued separately, not within explorations, and are compared in terms of query structure, not in terms of SQL operations w.r.t. previous queries. Thus, they capture users interests (e.g. which attributes are projected), not the way user navigates among data. The authors run their tests on smaller SQL workloads.

To our knowledge, this is the first attempt to learn human analysis behavior in SQL workloads.

2.3 Indicators for sequence analysis

Other research communities, in particular mobility science, study human behavior represented as sequences of actions. Data exploration can be viewed through the prism of mobility science [11]. Indeed, an exploration is a sequence of user's queries, where the movement is no longer conducted in space but in the *data space*.

Thus, many indicators proposed for the analysis of mobility sequences can be reused or adapted for the study of sequences of queries. Mobility researchers explored sequences of activities and tested the existence of simple universal rules underlying human movement like travel distance, top ranked visited locations, predictability of human activity and origin-destination flows,

Techniques	Description	Visual. method
	Statistical distribution	
Length distribution	Frequency distribution of sequence length in the dataset	Boxplot
State distribution	Frequency distribution of elements inside the sequences of the dataset	Barplot
	Vector description	
ℓ_1 norm	Sum of vector coordinates. $\|v\|_1 = \sum_{i=1}^{n} v_i$	Boxplot
Correlation	Correlation of vector dimensions in the dataset	Correlogram
Component analysis	Frequency of vector components	Barplot / Stackplot
	Transitions	
Origin-Destination matrix	Number of transitions from a vector q_i to q_j	Chord diagram
	Scattering and outliers	
UMAP	Dimensional reduction. Visualization of complex elements in 2D Euclidean spaces with a preservation of local topology	Euclidean projection

Table 1: Indicators for sequence and vector analysis

mainly studying recurring patterns/regularity in the sequence or clustering mobility behavior ([4] presents an important survey). In substance, results show that mobility is strongly characterized by exponential distribution (e.g. heavy-tailed, Zipf) and people constantly exploit a small set of repeatedly visited locations.

Inspired by these considerations, we propose to adapt a set of indicators from mobility mining to analyse data explorations. These complementary techniques, summarized in Table 1, highlight different aspects of explorations.

This capacity to explain models, both for practical and ethical issues, is a crucial point for the understanding of machine learning models. With this aim in mind, Guidotti and al. [10] suggested some techniques, partially borrowed from these above, like statistical methods and prototype selection elements, to explain black box systems in order to make their results more interpretable and understandable. In line with the vision of these techniques, we believe that the elaboration of indicators is essential to understand and explain discovered behavior in complex clusters.

2.4 Clustering methods

The extraction of behavior from a dataset is a process usually performed thanks to unsupervised machine learning. Indeed, clustering methods are widely used for the discovery of human behavior in datasets representing sequences of elements, in particular in sequences of mobility [14, 23, 27].

Clustering methods are based on similarity measures. A pairwise comparison of sequences results in a distance matrix that is the input of the clustering process. Many methods have been proposed for computing the similarity of categorical sequences. Most of the approaches are based on Optimal Maching (OM) methods [1], typical measures include those of the Edit Distance family (see [22] for a review of methods and similarity measures). In particular, the Contextual Edit Distance (CED) [23] is a generalization of Edit Distance, conceived for the comparison of semantic sequences (an overview of CED measure is given in Subsection 3.3).

However, the topology created by similarity measures for sequences is hard to apprehend. In particular, for OM methods, spaces are often not euclidean nor metric. To the best of our knowledge, the clustering algorithms able to deal with arbitrary distances (not necessarily metrics) are PAM [28] (or K-medoid), hierarchical clustering [15], density clustering (DBSCAN [9], OPTICS [3]) and spectral clustering [25], each one making different hypothesis about cluster topology.

According to the similarity measure and the representation of the sequences, dimensionality reduction methods can be used in order to extract primary dimensions [14]. However, commonly used methods like PCA can only be used for Euclidean spaces in practice. Alternatively, methods like UMAP [20], allow the reduction of a complex topology defined by an arbitrary metric into a low Euclidean space, which facilitates the visualisation of clustering results and enable the usage of other clustering methods, in particular, those requiring an Euclidean space like K-means [19]. In addition, UMAP offers a better preservation of the data global structure, fewer hyperparameters to tune and better speed than previous techniques like t-SNE [18].

In [22], we empirically compared several clustering methods and similarity measures, in order to find the most adapted to sequences of semantic elements. The combination of CED measure and UMAP reduction, paired with K-means, Spectral or DBSCAN algorithms, outperformed all other combinations of methods.

3 EXPLORATION MODEL

This section introduces the description of queries and explorations used all along the paper as well as their representation in a space of SQL operations.

The SQLShare workload contains 11,137 SQL statements, among which 10,668 correspond to SELECT statements. The remaining statements (mainly updates, inserts and deletes) were filtered. This workload was fragmented in 2,809 explorations containing among 1 and 98 queries [29].

3.1 Query and exploration abstractions

In what follows, we use the term *query* to denote the text of an SQL SELECT statement. In [29], queries are represented as a collection of fragments extracted from the query text, namely, projections, selections, aggregations and tables. We extend such representation adding group by and order by sets. These fragments abstract the most descriptive parts of a SQL query, and are the most used in the literature (see e.g., [8, 16, 21]). But note that we do not restrict to SPJG (selection-projection-join-group) queries. Indeed, we consider all queries in the SQLShare workload, some of them containing arbitrarily complex chains of subqueries.

Definition 3.1 (Query). A *query* over database schema DB is a 6-uple $q = \langle P, S, A, T, G, O \rangle$ where:

Id	Name	Abbrev.	Description	Computation
F_1	NAP	+P	Number of added projections	$F_1(q_k, q_{k-1}) = \|P_k - P_{k-1}\|$
F_2	NDP	-P	Number of deleted projections	$F_2(q_k, q_{k-1}) = \|P_{k-1} - P_k\|$
F_3	NAS	+S	Number of added selections	$F_3(q_k, q_{k-1}) = \|S_k - S_{k-1}\|$
F_4	NDS	-S	Number of deleted selections	$F_4(q_k, q_{k-1}) = \|S_{k-1} - S_k\|$
F_5	NAA	+A	Number of added aggregations	$F_5(q_k, q_{k-1}) = \|A_k - A_{k-1}\|$
F_6	NDA	-A	Number of deleted aggregations	$F_6(q_k, q_{k-1}) = \|A_{k-1} - A_k\|$
F_7	NAT	+T	Number of added tables	$F_7(q_k, q_{k-1}) = \|T_k - T_{k-1}\|$
F_8	NDT	-T	Number of deleted tables	$F_8(q_k, q_{k-1}) = \|T_{k-1} - T_k\|$
F_9	NAG	+G	Number of added group by expressions	$F_9(q_k, q_{k-1}) = \|G_k - G_{k-1}\|$
F_{10}	NDG	-G	Number of deleted group by expressions	$F_{10}(q_k, q_{k-1}) = \|G_{k-1} - G_k\|$
F_{11}	NAO	+O	Number of added order by expressions	$F_{11}(q_k, q_{k-1}) = \|O_k - O_{k-1}\|$
F_{12}	NDO	-O	Number of deleted order by expressions	$F_{12}(q_k, q_{k-1}) = \|O_{k-1} - O_k\|$

Table 2: Query features

(1) P is a set of expressions (attributes or calculated expressions) appearing in the main SELECT clause (i.e. the outermost projection). We deal with * wild card by replacing it by the list of attributes it references.
(2) S is a set of atomic Boolean predicates, whose combination (conjunction, disjunction, etc.) defines the WHERE and HAVING clauses appearing in the query. We considered indistinctly all predicates appearing in the outermost statements as well as in inner sub-queries.
(3) A is a set of aggregation expressions appearing in the main SELECT clause (i.e. the outermost projection).
(4) T is a set of tables appearing in FROM clauses (outermost statement and inner sub-queries). Views, sub-queries and other expressions appearing in FROM clauses are parsed in order to obtain the referenced tables.
(5) G is a set of expressions appearing in GROUP BY clauses (outermost statement and inner sub-queries).
(6) O is a set of expressions appearing in ORDER BY clauses (outermost statement and inner sub-queries).

Note that although we consider tables, selections, group by sets and order by sets occurring in inner sub-queries, we limit to the outermost queries for projections and aggregations, as they correspond to attributes actually visualized by the user. We intentionally remain independent of presentation and optimization aspects, specially the order in which attributes are projected (and visualized by the user), the order in which tables are joined, etc. All the queries we considered are supposed to be well formed, and so we do not deal with query errors.

Finally, an *exploration* is a sequence of queries of a user.

Definition 3.2 (Exploration). Let DB be a database schema. An exploration $e = \langle q_1, \ldots, q_p \rangle$ over DB is a sequence of queries over DB. We note $q \in e$ if a query q appears in the exploration e, and $exploration(q)$ to refer to the exploration where q appears.

3.2 Query features

For each query, we extract a set of simple features computed from the query text and its relationship with previous query in an exploration. The set of features is inspired from our previous work [6, 7, 24], which models OLAP queries as a set of features capturing typical OLAP navigation. It intends to capture the set of SQL operations that express one query w.r.t. the previous one (e.g. adding a projection, which means that a query projects an additional attribute w.r.t. the previous one).

Table 2 presents the considered features, where added (resp., deleted) indicates the modification made compared to the previous query. In their definitions, let $q_k = \langle P_k, S_k, A_k, T_k, G_k, O_k \rangle$ be the query occurring at position k in the exploration e over the instance I of schema DB. Features are computed comparing the query q_k to the previous query in the exploration e, $q_{k-1} = \langle P_{k-1}, S_{k-1}, A_{k-1}, T_{k-1}, G_{k-1}, O_{k-1} \rangle$. For the first query of e, i.e. q_1, we consider as predecessor the "empty" query $q_0 = \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$. All the features are defined for $k \geq 1$.

In what follows, we represent a SQL query in the space of query features, i.e. as a 12-dimensional vector, each position corresponding to one of the features $F_1 \ldots F_{12}$ described in Table 2. This representation is at the core of our proposal for computing the similarity between queries. It focuses in operations between queries and is independent of the underlying database, i.e. a given sequence of operations, even on different databases, will result in the same sequence of query vectors.

Definition 3.3 (Query vector). Let q be a query and q' its predecessor in an exploration. A *query vector* is a 12-dimensional vector $v = \langle v_1, \ldots v_{12} \rangle$ where $v_i = F_i(q, q')$.

Example 1. *Consider an exploration e_1 composed of 4 queries:*
q_1: SELECT species FROM All3col;
q_2: SELECT species FROM All3col WHERE longitude < 0;
q_3: SELECT species, longitude, latitude FROM All3col;
q_4: SELECT species, longitude FROM All3col ORDER BY species;

Vector for q_1, $\langle 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0 \rangle$, indicates an added projection (species) and an added table (All3col) w.r.t. the empty query. Vectors for q_2, q_3 and q_4 indicate the differences w.r.t. previous queries, an added selection (longitude <0), 2 added projections (longitude, latitude) with a deleted selection, and 1 deleted projection with an added order by attribute (species): $\langle 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 \rangle$, $\langle 2, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0 \rangle$, $\langle 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0 \rangle$, resp. □

As vectors are long and may have many 0-valued coordinates, we concisely represent them by listing the occurring operations (the ones not 0-valued) in the form "$\pm nL$", where $L \in \{P, S, A, T, G, O\}$ and $n \geq 1$ (omitted if 1). Letters and signs refer to features (as abbreviated in Table 2) and n represents feature magnitude. For instance, the queries of Example 1 can be noted +P+T, +S, +2P-S, -P+O, resp.

Remark that this representation captures both, the richness of the exploration in terms of query fragments (projections, selections, etc.), captured by the vector of the first query in the exploration (which is compared to an empty query), and the differences among consecutive queries, captured by the vectors of the

following queries. As a consequence, in some explorations, the norm of the first vector may be greater that those of the following ones. For instance, vectors of exploration 9 of the SQLShare workload are: +10P+T, +T-T and +S+T-T.

Finally, in some analyses in Section 5, we focus on the type of operation, disregarding the magnitude (e.g. how many projections are concerned) and sign (addition or deletion). To this end, we define aggregated 6-dimensional vectors (one dimension per type of operation). Analogously, they can be concisely represented with letters P,S,A,T,G,O.

Definition 3.4 (Aggregated vector). Let $v = \langle v_1, ...v_{12} \rangle$ be a query vector. An *aggregated vector* is a 6-dimensional vector $w = \langle w_1, ...w_6 \rangle$ where $w_i = 1$ if $(v_{2i-1} > 0)$ or $(v_{2i} > 0)$, 0 else.

Example 2. *Aggregated vectors for queries in Example 1 are:* $\langle 1, 0, 0, 1, 0, 0 \rangle$, $\langle 0, 1, 0, 0, 0, 0 \rangle$, $\langle 1, 1, 0, 0, 0, 0 \rangle$, $\langle 1, 0, 0, 0, 0, 1 \rangle$, *resp. In concise notation, they are sketched: PT, S, PS and PO.* □

3.3 Query and exploration similarity

We use cosine similarity for computing similarity between query vectors. This measure is well suited to compute the similarity between two vectors and is normalized in $[0, 1]$. In this way, it favors more the nature of SQL operations than their number. To deal with null vectors, which are frequent in the SQLShare dataset (see Section 4), we set border cases as follows: (i) two null vectors are considered identical (similarity is 1), and (ii) one null vector is considered as completely different from a non-null vector (similarity is 0). Formally, given two query vectors v and v', cosine similarity is calculated as follows:

$$\cos(v, v') = \begin{cases} 1 & \text{if } \|v\| = 0 \text{ and } \|v'\| = 0 \\ 0 & \text{if } \|v\| = 0 \text{ or } \|v'\| = 0 \\ \frac{v \cdot v'}{\|v\|\|v'\|} & \text{else} \end{cases} \quad (1)$$

In order to compare explorations, we pair CED measure with the cosine similarity measure among query vectors.

CED is a generalization of the Edit Distance, adapting cost computation to typical characteristics of semantic sequences. In particular, CED answers the following requirements: (i) edition cost depends on the similarity of nearby elements (the more similar and closer the elements, the lower the cost of operations), (ii) edition of repeated close elements has low cost, and (iii) similar and close elements can be exchanged with a low cost.

We describe CED computation as defined in [23] and tuned in [24]. Firstly, CED modifies the cost function γ of Edit Distance to take into account the local context of each element in the sequence. Consider contextual edit operations of the form $O = (o, e, q, k)$, denoting the operation $o \in \{\texttt{add}, \texttt{mod}, \texttt{del}\}$ on exploration $e = \langle q_1, ...q_n \rangle$ at index k by query q. Let $\mathcal{O}$ be the set of all possible contextual edit operations, the cost function $\gamma : \mathcal{O} \rightarrow [0, 1]$ is defined as:

$$\gamma(O) = 1 - \max_{i \in [\![1,n]\!]} \{sim(q_i, q) \times v_i(O)\} \quad (2)$$

where: *sim* is a similarity measure between two queries, computed as the cosine similarity of query vectors, and $v(O) \in [0, 1]^n$ is a contextual vector which quantifies the notion of proximity between queries. Usually, bigger $|i - k|$ is, lesser $v_i(O)$. As in [24], we use: $v_i(O) = \exp\left(-\frac{1}{2}\left(\frac{2\sqrt{k+1}(i-k)}{|e|}\right)^2\right)$

CED is computed as Edit Distance, using dynamic programming and Wagner-Fisher algorithm [36].

In next sections we describe how this representation of queries and operations is used for profiling the SQLShare workload and clustering explorations.

4 DATASET PROFILING

The SQLShare workload contains 2,809 explorations, totalizing 10,668 queries. Length of explorations follows the Zipf's law. Indeed, 1,379 explorations are one-shot (i.e. they contain only one query), median exploration contains 2 queries and the longest one contains 98 queries. Figure 1a shows the boxplot of the distribution.

The number of operations in a query (w.r.t. previous query) also follows the Zipf's law. Noticeably, 1,289 out of 10,668 queries have no operations w.r.t. previous query. This happens when a query is identical to previous one (947 queries) but also when there are only visualisation or optimisation changes (e.g. changing the order of projected attributes or joined tables) and when changes concern advanced options not captured by our features (e.g. changing a regular join by an outer join, or changing the ascending/descending sense of ordering). Mean query has 4 operations and the longest one has 510 operations. The latter correspond to a query of the form "SELECT * FROM T" where T contains a large number of columns. Figure 1b shows the boxplot of the distribution. We notice that many outliers correspond to large first queries (i.e. containing many fragments, specially projections) which lead to long query vectors when compared to the empty query q_0.

Most frequent operations are adding and deleting projections (+P and -P), adding tables (+T) and adding selections (+S). Less frequent operations concern group by (+G and -G) and order by (+O and -O). Figure 1c shows the complete distribution.

Figures 1d and 1e complement the distribution of operations by highlighting the combinations of operations that are frequently performed together. Precisely, Figure 1d shows the top 10 most frequent query vectors, evidencing that null vectors ($\emptyset$) are the most frequent, followed by a change in selections (+S-S) and a change in projections (+P-P). Some frequent vectors are surprising, as changing one table without changing anything else (+T-T). This behavior corresponds to users updating and uploading a dataset and then evaluating the same query in the new dataset. In addition, Figure 1e shows the top 10 most frequent aggregated query vectors, and consequently illustrating the most frequent types of operations disregarding vector magnitude and sign. Most frequent aggregated vectors concern changes in projections, selections and tables (PST) and subsets of these operations (PT, P and S). Interestingly, null vectors ($\emptyset$) come in fifth position. These top 10 aggregated vectors cover 9,205 queries (82.3% of the total number of queries).

Figure 1f goes a step forward showing the main flows[2] between aggregated query vectors, by means of a Chord diagram. A flow $\langle A, B \rangle$ indicates queries with vector A followed by queries with vector B. It is represented by an arrow (the origin being closer to the external circle), whose magnitude indicates its frequency. For example, the flow from *PST* to *S* (in purple) represent *PST* vectors followed by *S* vectors. We can observe that many autoflows (e.g. for *S*, *P*, *PST*). Interestingly, many null vectors ($\emptyset$) are followed by other null vectors or by simple changes (*P* and *S*).

[2]Main flows are the ones such that the number of transitions is greater than 5% of the biggest flow.

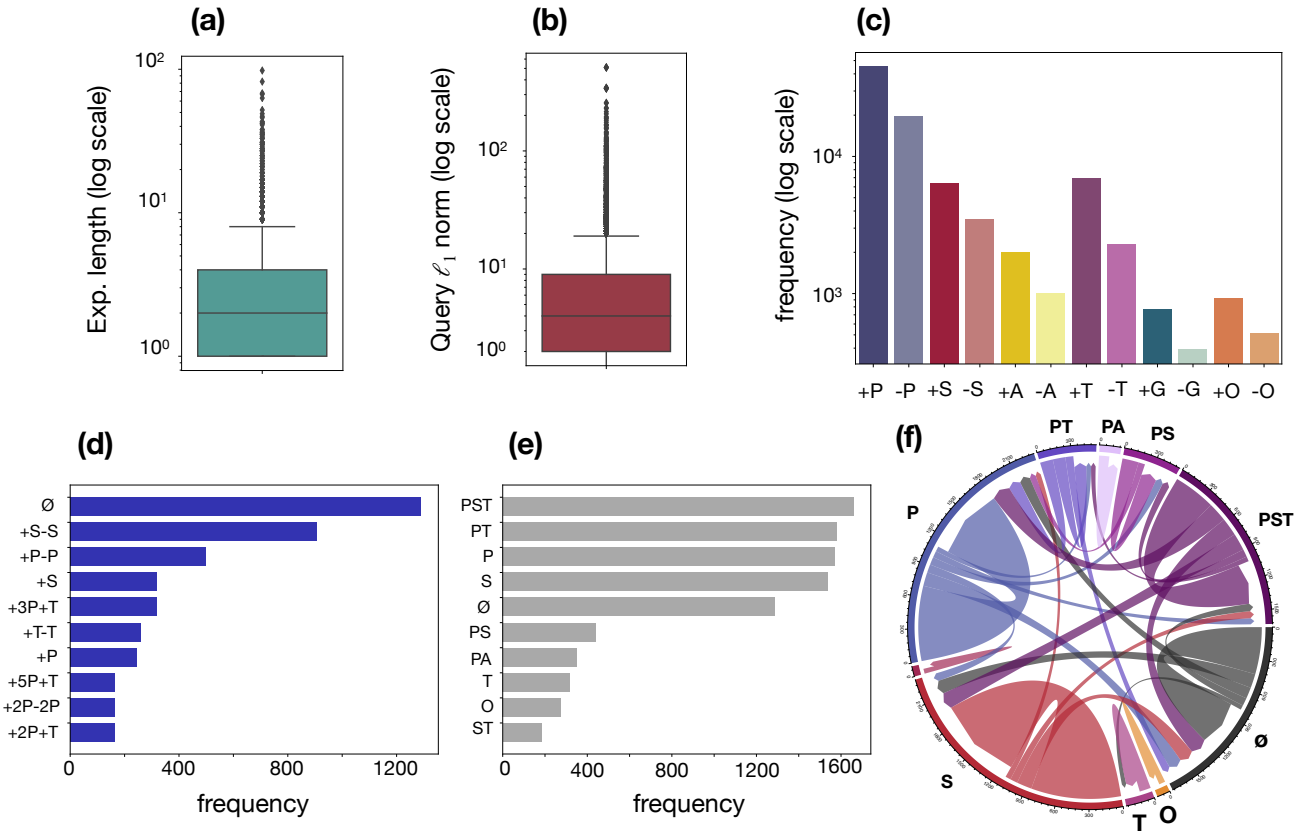

Figure 1: Dataset profiling: (*a*) Boxplot of exploration lengths, (*b*) Boxplot of number of operations in queries (ℓ_1 norm), (*c*) Frequency distributions of the operations, (*d*) Top 10 query vectors, (*e*) Top 10 aggregated query vectors, and (*f*) Flows between aggregated vectors

5 CLUSTERING OF EXPLORATIONS

This section presents our proposal for clustering explorations, defining the experimental protocol and implementation setting. We also present a large analysis of the obtained clusters, explaining the analysis behaviors they represent.

5.1 Experimental protocol

Based on the empirical results of [22], we use the combination of CED [23], UMAP [20] and DBSCAN [9], that best performed on sequences of semantic elements. Indeed, as will be shown later in this section, DBSCAN is well suited to the topology resulting from CED and UMAP when applied on the SQLshare workload.

5.1.1 Implementation and settings. We used the CED implementation and setting described in [24], which is recalled in Subsection 3.3. Concerning UMAP, we use the *umap-learn* python library 0.4.3, where *min_dist* is set to 0.01, *n_neighbors* to 200 (i.e. around 10% of the dataset) and pseudo random number generator seed is 42. Finally, according to the UMAP projection, we use the DBSCAN clustering algorithm from the *sklearn* python library 0.22.2, applied on the previous UMAP embedding, with $eps = 1$ and $min\_samples = 10$.

All experiments are available and can be reproduced by running our Python notebook[3] in Google Colab or Jupyter environments. In particular, all code generating the graphs, the dataset profiling, the clustering analysis and further experiments are available in the notebook.

[3]https://colab.research.google.com/drive/1Yt7Q7AFghkcxdea2UicccMCmkaX7dRMD?usp=sharing

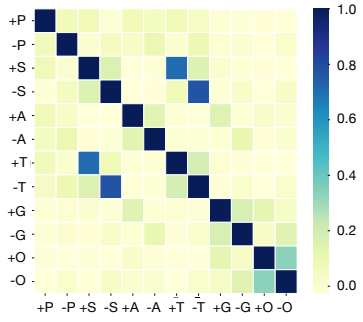

Figure 2: Correlation matrix between all query features

5.1.2 Protocol description. To further justify our choice of methods, we performed some preliminary tests on workloads of explorations having a ground truth. Specifically, we used the workloads of OLAP explorations described in [24], and obtained comparable results for the artificial dataset and improved results for the explorations of real users (Ipums dataset). These results are available in the notebook; we omit them here for lack of space. Remark that as features are lowly correlated (correlation matrix is shown in Figure 2), we decided to keep all of them. A PCA analysis confirmed this choice.

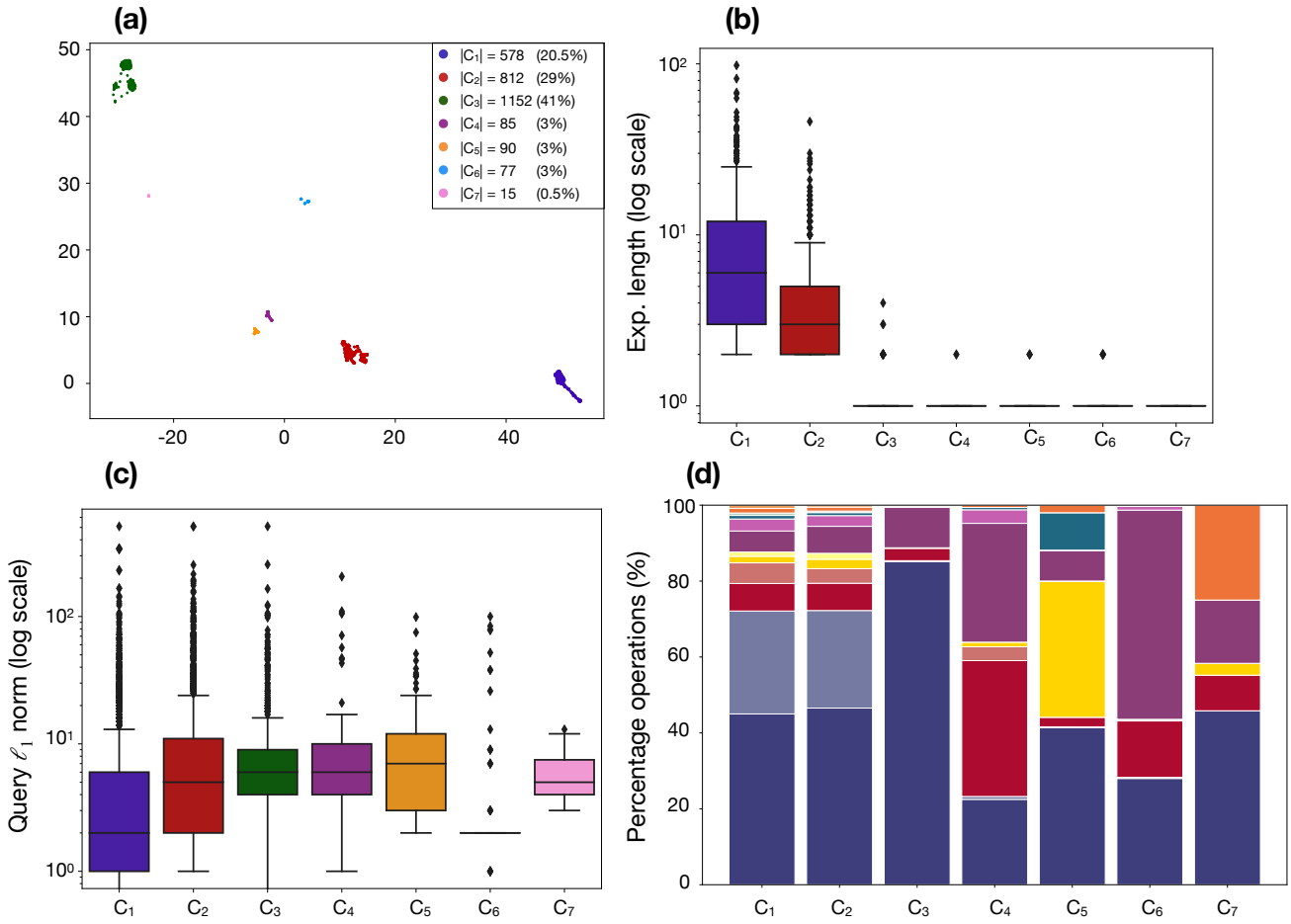

Figure 3: Whole clustering: (*a*) UMAP Reduction and DBSCAN partition (cluster sizes in legend), (*b*) Boxplots of exploration lengths, (*c*) Boxplot of number of operations in queries (ℓ_1 norm) (*d*) Stack plot of the distribution of the operations in each cluster (colors are the ones used in Figure 1c)

Finally, given the large number of one-shot explorations, as shown in Figure 1a), we decided to test two clustering configurations: (i) *whole clustering* (on the whole dataset), and (ii) *restricted clustering* (excluding one-shot explorations). Indeed, the unique query of such explorations, when compared to the empty query (q_0), are 0-valued for features $F_2, F_4, F_6, F_8, F_{10}$ and F_{12} (which count the deleted query fragments), introducing a bias. The restricted clustering aims to further analyse longer explorations, revealing richer patterns.

Later in the section, we describe the results of both clustering configurations.

5.2 Results of the whole clustering

The clustering of the whole dataset resulted in 7 clusters. Figure 3a plots the UMAP reduction of the dataset to a 2D Euclidean space. Clusters had varying sizes. Indeed, there are 3 large clusters (C_1 to C_3), which explorations exhibit frequent behavior, and 4 small clusters (C_4 to C_7) concerning less frequent behavior.

As expected, the length of explorations had a big impact in clustering results. As shown in Figure 3b, clusters C_1 and C_2 contain only explorations having at least 2 queries while the remaining clusters contain a majority of one-shot explorations. Indeed, clusters C_4 to C_6 contain some explorations of length 2, and cluster C_3 contains some longer ones. Cluster C_7 only contains one-shot explorations. On average, cluster C_1 contains longer explorations than C_2, including the longest ones.

The overall distribution of operations for clusters C_1 and C_2 is very similar (see Figure 3d), however, clusters differentiates in the number of operations among consecutive queries (captured by the ℓ_1 norm of query vectors, shown in Figure 3c) and in their flows. Although explorations in cluster C_1 are longer than those in cluster C_2, they have less operations (median=2). Furthermore, the most frequent aggregated query vectors, listed in Table 3, confirm this observation. In particular, aggregated vectors in C_1 include most of the null vectors, but also many vectors representing only one type of operation (esp. *S* and *P*). However, many frequent vectors in cluster C_2 concern many operations. We further analyse these two clusters in next subsection.

Cluster C_3 is the largest one and contains a majority of one-shot explorations. Queries in its explorations contain many operations (median=6), in majority projections. There are two frequent aggregated vectors: *PT* and *PST*.

Clusters C_4 to C_7 are smaller, contain mostly one-shot explorations, but evidencing very different behavior. Queries in cluster C_4 involve many operations (median=6), concerning many selections and tables. The most frequent aggregated vector is *PST*. Queries in cluster C_5 also involve many operations (median=6), concerning many projections and aggregations, some grouping

	Clust id	C_1	C_2	C_3	C_4	C_5	C_6	C_7
Rank	1	∅ (1288)	P (736)	PT (775)	PST (62)	PAT (29)	PT (69)	PTO (10)
	2	S (1070)	PST (627)	PST (319)	PSAT (9)	PATG (28)	T (6)	PSATO (2)
	3	P (787)	S (467)	P (48)	PSTO (5)	PATGO (15)	PST (5)	PSTO (2)
	4	PST (645)	PT (441)	PSTO (13)	PSATG (4)	PSATGO (7)	PAT (2)	PSO (1)
	5	PT (296)	PS (225)	PTO (6)	PSATGO (3)	PSTG (5)	PSTO (2)	
	6	PA (210)	PA (137)	PATO (3)	PS (3)	PSATG (5)		
	7	T (187)	T (126)	PTG (3)	S (1)	PA (2)		
	8	O (187)	O (94)	∅ (1)		PG (2)		
	9	PS (176)	PSTO (81)	PSO (1)		PSTGO (1)		
	10	ST (119)	PAT (75)					
Cover representation		88%	72%	100%	100%	100%	100%	100%

Table 3: Top 10 frequent aggregated query vectors with frequency (in brackets)

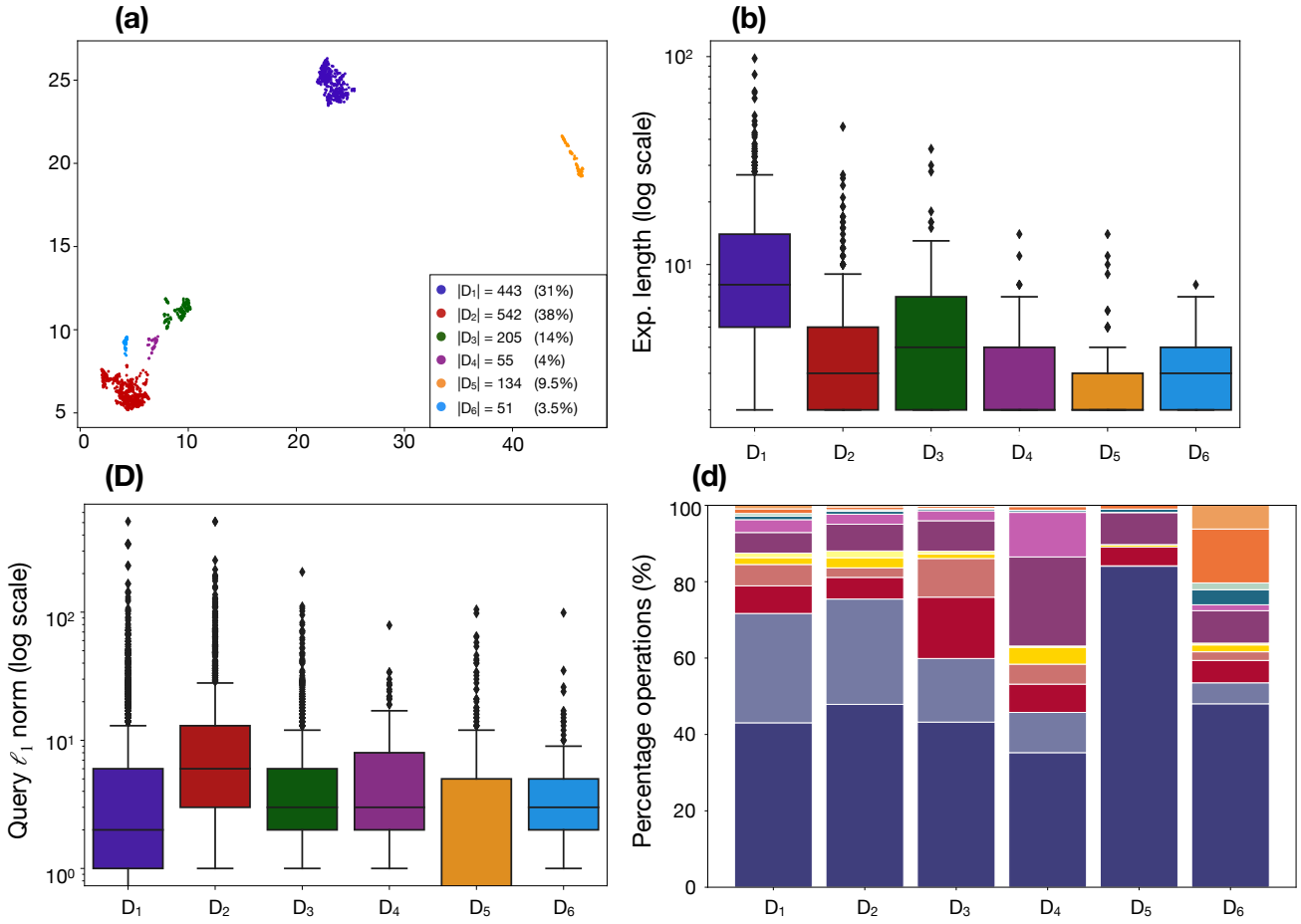

Figure 4: Restricted clustering: (*a*) UMAP Reduction and DBSCAN partition (cluster sizes in legend), (*b*) Boxplots of exploration lengths, (*c*) Boxplot of number of operations in queries (ℓ_1 norm) (*d*) Stack plot of the distribution of the operations in each cluster (colors are the ones used in Figure 1c)

but few selections. The most frequent aggregated vectors are *PAT* and *PATG*. There are two types of queries in cluster C_6, as evidenced in Figure 3c. Most queries involve only 2 operations (+P+T), the others concern multiple operations (multiple tables and many projections). The most frequent aggregated vector is *PT*. Cluster C_7 is the smaller one (only 15 explorations). Its queries have fewer operations, concerning many projections and ordering. The most frequent aggregated vector is *PTO*.

5.3 Results of the restricted clustering

The restricted clustering resulted in 6 clusters, plot in Figure 4a. There are two well differentiated clusters (D_1 and D_5) and a dense zone including a large cluster (D_2) and 3 smaller ones (D_3, D_4 and D_6). It is here, that DBSCAN best exploits the space topology. Cluster analysis is shown in Figure 4 and the main flows (for the bigger clusters) are shown in Figure 5.

Explorations coming from cluster C_1 are distributed between cluster D_1 and D_5 (excepting 1 exploration that goes to cluster

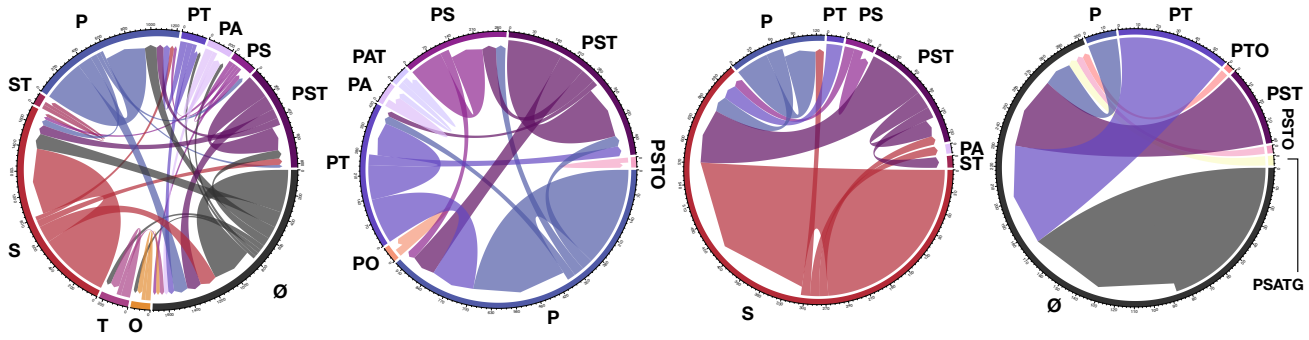

Figure 5: Flows in main restricted clusters. Left to right: D_1, D_2, D_3 and D_5

	Cluster	Median exp. length	Median nb. op.	Common op.	Freq. agg. vect.	Workload coverage	Pattern nickname
One-Shot	C_3	1	6	+P +T	PT, PST	41%	*Full PROJECTIONS*
	C_4	1	6	+S +T +P	PST	3%	*FILTER enthusiast*
	C_5	1	7	+P +A +G	PAT, PATG	3%	*AGGREGATOR*
	C_6	1	2	+T +P	PT	3%	*Table JOINER*
	C_7	1	5	+P +O +T	PTO	< 1%	*Ordered*
Longer exp.	D_1	8	2	+P -P +S	∅, S, P, PST	16%	*Long & Focused*
	D_2	3	6	+P -P +T	P, PST, PS, PT	19%	*PROJECTION chains*
	D_3	4	3	+P +S -P -S	S, PST, P	7%	*FILTER chains*
	D_4	2	3	+P +T -T	T, PST, PT	2%	*Dataset reloader*
	D_5	2	0	+P +T +S	∅, PT, PST	5%	*Repeater*
	D_6	3	3	+P +O +T	O	2%	*ORDER maniac*

Table 4: Summary of learned behavior: median exploration length, median number of operations per query, common operations, frequent aggregated vectors and workload coverage

D_2). Cluster D_1 contains the longest explorations and the highest median number of explorations. Its queries concern operations of varied types, most of them limiting to 1 or 2 operations. Frequent aggregated vectors are ∅, *S*, *P* and *PST*. Flows illustrates many repetitions of the same operations but also the alternation of operations. Cluster D_5 contains shorter explorations. 62 % of queries are identical to previous ones (empty vector). In the remaining ones, projections are predominant, with some selections and tables. Frequent aggregated vectors are ∅, *PT* and *PST*. Flows evidence that first queries in the explorations have typical vectors (e.g. *PT*, *PST*) and next queries are identical (∅).

Explorations coming from cluster C_2 are distributed among clusters D_2, D_3, D_4 and D_6, in the dense zone. In addition, the 40 explorations coming from the other clusters (C_3 to C_6) mainly goes to D_2. The four clusters contain, in average, shorter explorations than cluster D_1, with more operations per query. Queries in cluster D_2 concern operations of varied types, most of them being projections. Frequent aggregated vectors are *P*, *PST*, *PT* and *PS*. Flows evidence that explorations concern chains of the same operations (visible in the autoflows). There are more selections in queries of cluster D_3, frequent aggregated vectors being *S*, *PST* and *P*. Flows show that first queries (mainly with vectors *PT*, *PS* and *PST* are followed by chains of selections, and some marginal projections. Queries in cluster D_4 concern many changes in tables, and more aggregations that previous clusters. Frequent aggregated vectors are *T*, *PT* and *PST*. Finally, queries in cluster D_6 concern most of the order by operations, but all types of operations are present. The most frequent aggregated vector is *O*.

5.4 Learned behavior

The analysis of both clustering configurations allowed the discovery of several patterns, representing common or less-frequent behavior. The most prominent aspects of each pattern are summarized in Table 4. This section briefly highlights our findings.

Firstly, 49% of explorations are one-shot. They differentiate in the predominant operations in the unique query. We discovered 5 patterns. The most common one (C_3) consist in evaluating a simple query, projecting many attributes, possibly to verify that the dataset was correctly uploaded or just looking at the data.

Less frequent patterns, also concerning the evaluation of a simple query, differentiate in the used SQL operations, namely, many selections (C_4), aggregation and grouping (C_5), join of multiple tables (C_6), and ordering (C_7). The latter is an outlier behavior, only concerning 15 explorations. These patterns suggest a more specific analysis of data (w.r.t. the common behavior in C_3), taking advantage of more SQL operations. This may reflect users' preferences on some SQL clauses, but may also reflect users' expertise.

The remaining 51% of explorations contain between 2 and 98 queries, median being 4 queries. We discovered 6 patterns:

A common pattern (D_1) reveals long explorations, with few operations per query, sometimes repeating queries, which translate a focused data analysis. Many types of operations are used, but mostly once per query, suggesting a conscious use of SQL.

Another common pattern (D_2) reveals short explorations, with more operations per query. Projections are omnipresent, but frequently combined with other operations. What is interesting here,

is the chaining of the same types of operations along the exploration. It can be exploited for providing personalized suggestions to users.

Two interesting but less frequent patterns (D_3 and D_5) concern a classical first query, followed by chains of selections (D_3) or repeated queries (D_5). In both cases, explorations are shorter than in D_1 but reveal some kind of analysis. While D_3 suggest a meticulous study of the dataset, D_5 includes many novices users trying to understand how SQL works. A similarly but more complex pattern (D_6) involve more richer first queries, followed by changes in the ordering of projected expressions. In addition to a good use of SQL, this behavior may correspond to users looking for the best way of reporting data.

The last pattern (D_4), also less frequent, exhibits a particular behavior. It concerns many changes in the datasets (frequently, the unique operation in the query is a change in the FROM clause). This corresponds to the upload of a new dataset and the execution of the same query on the new dataset, and suggests data analysts dealing with quality issues in their datasets.

6 CONCLUSION AND FUTURE WORKS

This paper presented an original solution to learn analysis behavior in SQL workloads. The understanding of users' analysis patterns has great implications for query recommendation, monitoring, optimization and, more generally, providing better IDE support. The proposal includes an abstraction of queries and explorations in the space of SQL operations, a set of similarity functions tailored for SQL queries and explorations, and an innovative clustering process taking advantage of UMAP reduction for analysing a complex space.

The approach was tested on a real workload, SQLShare, allowing the extraction of 11 analysis patterns including 3 typical behaviors: one-shot simple explorations, short exploratory explorations, and longer more focused ones, but also less-frequent behavior evidencing the punctual use or the chaining of specific SQL operations. We believe that the identification of such behavior should be at the kernel of more intelligent IDE tools.

In this paper we used a large palette of indicators for profiling the workload and analyzing the obtained clusters (some additional ones are described in our notebook). In next future, we would like to test additional indicators, specifically concerning how focused are the explorations (i.e. distinguishing flows at the beginning and end of explorations), and how complex are queries, both in terms of expressiveness and usage of advanced clauses and functions (here, we also need to extract additional features). In addition, we would like to classify users according to their analysis behaviors. SQLShare workload, with its 57 users and their 3,336 datasets, is a rich source for further experiments. Of course, there are many one-shot users (already reported in [13]), but our preliminary analyses reveal very interesting behavior.

Finally, we would like to test our proposal in further workloads, specially those including queries generated by bots, as SDSS. Authors of [32] acknowledge the difficulty of extracting human sessions from all those collected: *"We failed to find clear ways to segment user populations. [...] Interactive human users were 51% of the sessions, 41% of the Web traffic and 10% of the SQL traffic. We cannot be sure of those numbers because we did not find a very reliable way of classifying bots vs mortals."* Developping tools helping in the recognition and analysis of hand-written queries is a nice challenge.

REFERENCES

[1] A. Abbott and A. Tsay. Sequence analysis and optimal matching methods in sociology: Review and prospect. *SMR*, 29(1):3–33, 2000.
[2] A. C. Acar and A. Motro. Why is this user asking so many questions? explaining sequences of queries. In *DBSec*, 2004.
[3] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander. Optics: ordering points to identify the clustering structure. *ACM Sigmod*, 28(2):49–60, 1999.
[4] H. Barbosa, M. Barthelemy, G. Ghoshal, C. R. James, M. Lenormand, T. Louail, R. Menezes, J. J. Ramasco, F. Simini, and M. Tomasini. Human mobility: Models and applications. *Physics Reports*, 734:1 – 74, 2018.
[5] S. Chaudhuri and V. R. Narasayya. Self-tuning database systems: A decade of progress. In *VLDB*, 2007.
[6] M. Djedaini, K. Drushku, N. Labroche, P. Marcel, V. Peralta, and W. Verdeaux. Automatic assessment of interactive OLAP explorations. *Inf. Syst.*, 82:148–163, 2019.
[7] M. Djedaini, N. Labroche, P. Marcel, and V. Peralta. Detecting user focus in OLAP analyses. In *ADBIS'2017, Nicosia, Cyprus*, 2017.
[8] M. Eirinaki, S. Abraham, N. Polyzotis, and N. Shaikh. Querie: Collaborative database exploration. *TKDE*, 26(7):1778–1790, 2014.
[9] M. Ester, H.-P. Kriegel, Sander, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. *Kdd*, 96(34):226–231, 1996.
[10] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi. A survey of methods for explaining black box models. *ACM CSUR*, 51(5), 2018.
[11] T. Hägerstraand. What about people in regional science? *Papers in regional science*, 24(1):7–24, 1970.
[12] S. Idreos, O. Papaemmanouil, and S. Chaudhuri. Overview of data exploration techniques. In *SIGMOD*, 2015.
[13] S. Jain, D. Moritz, D. Halperin, B. Howe, and E. Lazowska. Sqlshare: Results from a multi-year sql-as-a-service experiment. In *SIGMOD*, 2016.
[14] S. Jiang, J. Ferreira, and M. C. González. Clustering daily patterns of human activities in the city. *DMKD*, 25(3):478–510, 2012.
[15] L. Kaufman and P. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, 2009.
[16] N. Khoussainova, Y. Kwon, M. Balazinska, and D. Suciu. Snipsuggest: Context-aware autocompletion for SQL. *PVLDB*, 4(1):22–33, 2010.
[17] G. Kul, D. T. A. Luong, T. Xie, V. Chandola, O. Kennedy, and S. J. Upadhyaya. Similarity metrics for SQL query clustering. *IEEE Trans. Knowl. Data Eng.*, 30(12):2408–2420, 2018.
[18] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9:2579–2605, 2008.
[19] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297, 1967.
[20] L. McInnes, J. Healy, and J. Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.
[21] T. Milo and A. Somech. Next-step suggestions for modern interactive data analysis platforms. In *KDD*, 2018.
[22] C. Moreau, A. Chanson, V. Peralta, T. Devogele, and C. de Runz. Clustering sequences of multi-dimensional sets of semantic elements. In *SAC*, 2021.
[23] C. Moreau, T. Devogele, V. Peralta, and L. Étienne. A contextual edit distance for semantic trajectories. In *SAC*, 2020.
[24] C. Moreau, V. Peralta, P. Marcel, A. Chanson, and T. Devogele. Learning analysis patterns using a contextual edit distance. In *DOLAP*, 2020.
[25] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in NIPS*, pages 849–856, 2002.
[26] H. V. Nguyen, K. Böhm, F. Becker, B. Goldman, G. Hinkel, and E. Müller. Identifying user interests within the data space - a case study with skyserver. In *EDBT*, 2015.
[27] L. Pappalardo, F. Simini, S. Rinzivillo, D. Pedreschi, F. Giannotti, and A.-L. Barabási. Returners and explorers dichotomy in human mobility. *Nature communications*, 6(1):1–8, 2015.
[28] H.-S. Park and C.-H. Jun. A simple and fast algorithm for k-medoids clustering. *Expert systems with applications*, 36(2):3336–3341, 2009.
[29] V. Peralta, P. Marcel, W. Verdeaux, and A. S. Diakhaby. Detecting coherent explorations in SQL workloads. *Inf. Syst.*, 92, 2020.
[30] S. Rizzi and E. Gallinucci. Cubeload: A parametric generator of realistic OLAP workloads. In *CAiSE*, 2014.
[31] O. Romero, P. Marcel, A. Abelló, V. Peralta, and L. Bellatreche. Describing analytical sessions using a multidimensional algebra. In *DaWaK'2011, Toulouse, France*, 2011.
[32] V. Singh, J. Gray, A. Thakar, A. S. Szalay, J. Raddick, B. Boroski, S. Lebedeva, and B. Yanny. Skyserver traffic report - the first five years. Technical report, December 2006.
[33] A. Somech, T. Milo, and C. Ozeri. Predicting "what is interesting" by mining interactive-data-analysis session logs. In *EDBT*, 2019.
[34] H. van den Brink, R. van der Leek, and J. Visser. Quality assessment for embedded SQL. In *SCAM*, pages 163–170. IEEE Computer Society, 2007.
[35] A. Vashistha and S. Jain. Measuring query complexity in sqlshare workload. https://uwescience.github.io/sqlshare/pdfs/Jain-Vashistha.pdf.
[36] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, 1974.
[37] R. W. White. *Interactions with Search Systems*. Cambridge Univ. Press, 2016.