SQLVis: Visual Query Representations for Supporting SQL Learners

SQLVis: Visual Query Representations for Supporting SQL Learners

Daphne Miedema, George Fletcher
Department of Mathematics and Computer Science
Eindhoven University of Technology
{d.e.miedema, g.h.l.fletcher}@tue.nl

***Abstract*—SQL is a typical query language for performing data analytics. Although its usage is ubiquitous, learners experience that query formulation in SQL is error-prone and time-consuming. Prior research has shown that this is due to low expressive ease, extensive training requirements and high cognitive load, all of which present a significant burden for SQL learners. Visual representations can assist learners to significantly lower this burden. The current dominant paradigm aims to facilitate SQL querying by helping users to avoid the syntax of SQL. Such Visual Querying Systems (VQS), however, are not effective for SQL learners as they hide the syntax of the language during query formulation, rather than assisting learners to write correct queries in SQL. Furthermore, training with VQSs is system specific, which leads to system dependency for learners. We argue that novices need support from Visual Query Representation (VQR) solutions which, instead, help them in learning how to write correct and portable SQL queries. In this paper we present SQLVis, a VQR to support novice SQL users in query writing. Our system represents the query as written in SQL by the user, which can improve the SQL writing proficiency of its users. Results of an in depth empirical study demonstrate the significant value of SQLVis for learners.**

***Index Terms*—Query languages, Visualization techniques and methodologies, Computer Science education**

I. Introduction

Data is commonly stored in relational databases. The most popular language through which this data can be queried is the Structured Query Language (SQL). Previous research has shown that using SQL is difficult and error-prone [1], [10], [21]. Extensive training is required before one can use SQL effectively. In addition, the cognitive load of writing queries is high [4], [28]. There are various aspects that may play a role, for example: keeping the database schema in working memory, the system only catching certain types of errors, low expressive ease, and losing track of your query while writing it [11], [28], [31]. These issues occur especially for novices, which hinders the process of learning SQL.

Despite the high barrier to entry, SQL is ubiquitous [37], in education and practice. To simplify the querying process, many researchers have presented methods that allow for visual query building, such that no knowledge of syntax is required for writing queries in SQL [22], [30], [35], [39]. Such Visual Querying Systems (VQS) generate a query from a visual representation constructed by the user. A VQS simplifies interactions with the database by abstracting away from the language syntax, which leads to increased information throughput [8]. However, they also increase system dependency, as a VQS obfuscates syntax instead of supporting the learning of SQL.

978-1-5386-5541-2/18/$31.00 ©2021 IEEE

```
SELECT MAX(quantity),
       p.pName
FROM purchase AS pur,
     product AS p
WHERE p.pID = pur.pID
AND pur.price > 10;
```

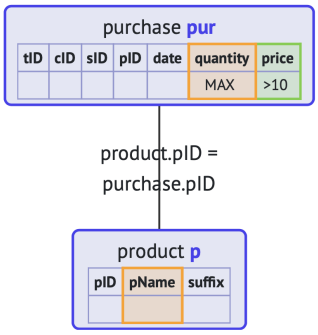


Fig. 1: A user query and its corresponding SQLVis visualization. Tables are encoded as nodes, and constraints that link tables together as edges. The returned attributes are highlighted in orange, column based contraints in green.

Another approach to using visualizations to support query formulation is to generate a representation from a written query. This can either be done by annotating the result table, or in the form of a separate visualization. We call the former Visual Result table Representations (VRR), and various such systems exist [9], [14], [23], [29]. However, this approach does not scale well for queries involving many (or large) tables, or queries with many results. In contrast, a Visual Query Representation (VQR) is a visualization that represents the *query*, without taking into account the result table. This has the advantage that the representation is relatively small and more structured than a representation on the result table. Figure 2 contains an overview of the three types of visual tools.

The VQR approach of query formulation and post-hoc visualization, facilitates the learning of the query language. In addition, a VQR supports the learner by visualizing the meanings of SQL commands, keeping track of query formulation, and relieving the user of the burden of keeping the database schema in memory. Juxtaposing the representation and the query can help the user understand the effect of using various SQL clauses. So, contrary to the VQS approach, with a VQR we can support the learner, while also familiarizing them with SQL. For learners, a VQR is also preferable over a VRR as the former is a more direct and compact representation. Through all these advantages, a VQR can be a great aid in database education and training.

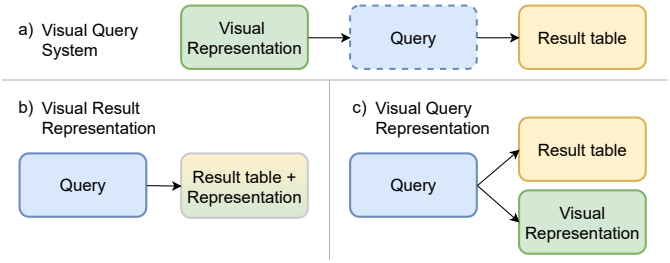


Fig. 2: (a) In a VQS, query text is implicit. (b) A VRR is generated from user-defined SQL queries, and shows a visual representation as a highlight on, or conjuction of, (intermediate) result tables. (c) Similar to b, a VQR represents the user-defined query, but do this disjoint from the result table.

VQR is a relatively new and novel approach, representing a paradigm-shift from user to learner. To our knowledge, QueryVis is the only prior VQR work [25]. In contrast to supporting learners during query formulation, however, QueryVis focuses on interpretation of existing queries. This leads to a difference in design decisions that we present in Section III.

Our contributions. In this paper we present SQLVis[1], a Visual Query Representation for SQL learners. SQLVis leverages a graph-based query representation. A graph structure for VQR increases understanding by adding information, as SQLs implicit relations in the data (through foreign keys) can be confusing. SQLVis makes these foreign key relations explicit, which lowers cognitive load.

We proceed first with grounding the design of SQLVis in the literature on SQL- and graph visualization. When then analyze SQLVis with respect to two research questions:

Q1 How do users assess SQLVis?
Q2 What is the effect of using SQLVis on the query formulation process?

We evaluate **Q1** and **Q2** with a qualitative and a quantitative study, which show that SQLVis is evaluated positively and query formulation through SQLVis leads to fewer mistakes than query formulation without SQLVis. We conclude with pointers for future research building upon our contributions.

II. Related work

A. Visual representations and cognitive load.

In this section we give an overview of reasons why adding a visual representation to a system can be beneficial. First of all, a visual representation provides the opportunity to apply parallel processing. This is possible due to the fact that short-term memory gathers information through two separate channels within working memory: the visuo-spatial sketchpad and the phonological loop. The first presents the working memory with visual input, the second propels verbal and textual information. Both these information channels and the working memory as a whole have a limited capacity. The amount of working memory resources in use is called the cognitive load. The concept that the chosen learning method influences cognitive load was posed by Sweller [36].

[1]https://github.com/Giraphne/sqlvis

Queries in SQL are purely textual, processed in the phonological loop. By adding a visual representation in the interface, some load can be taken off the phonological loop by redistributing it to the visuo-spatial sketchpad. This dual-coding of information has a memory-enhancing effect through freeing of resources [13], [20], [33]. This holds especially in cases where the textual and visual data are highly related [19], close together and present new information [33].

There are various ways in which a representation can support data analysis: through parallel processing, as external memory, and by imposing structure on the data [40]. Yung et al. found that a visual representation helped their participants focus on their attention on the most essential elements [42].

Visual representations can also aid the development of appropriate mental models. Schnotz et al. found evidence of a correspondence between the structure of a graphical representation and its associated mental model [33]. There also seems to be an effect of experience: as their knowledge is typically fragmented [13] novices benefit from pictures in text [33]. Much of the information presented to an expert is already in their long-term memory and thus will not overwhelm them [13]. However, incorrect pictures may confuse experts, thus a representation needs to be appropriately designed [33]. The integration of new information with existing knowledge is an essential part of meaningful learning [26].

B. Representing SQL queries.

As research has shown that one of the main problems in query formulation is high cognitive load [4], [28], SQL can profit from this effect of visual representations. Indeed, recent work has underscored the importance of further research on visual support for query formulation [16]. High cognitive load leads to errors in query formulation, with specific SQL keywords that users make the most mistakes in. Examples of difficulty-inducing concepts are aggregate functions such as AVG, COUNT and MAX [7], [37], grouping constructs such as GROUP BY and HAVING [1], [7], [23], [28], [37], and concepts such as subqueries and negation [27], [37].

Subqueries can be difficult to understand due to their intrinsic nested nature. Queries can contain many levels of nesting, making them hard to write, and even harder to understand. For the visual representation of subqueries, there is a clear-cut answer: delimiting each level of nesting and each subquery within that with a box, nesting the boxes as the queries are nested. This is supported by earlier research, such as [38]. Another method is to represent nesting through directed arrows, which indicate a reading order [17].

There is less consensus on the optimal way to represent negation. It is relatively easy to show the presence of a concept, but how does one show something that must not be there? Earlier work suggest the use of complementary colors, distinguishing between existing and not existing [24]. Others suggest different borders for different types of complex operations (including conjunction and disjunction) [5], [17]. The most primitive form of negation through colors and borders is the notion of a ‘cut’, as introduced by Peirce in the 19th century [12]. Finally, negation can also be expressed through the use of symbols such as $\neg$ or $\sim$ [12].

There are various aggregate functions available in SQL. All correspond to mathematical functions that can be evaluated on the data. They are always called on one or more columns in the data, and thus it would make sense to highlight them there. Thalheim developed Visual SQL [38], a representation of SQL as boxes and connections. They define aggregation elements as algebraic expressions that are added to the unit on which the expression is applied. Similarly, Leventidis et al present aggregates in the column of the schema that they are applied on [25], similar to how they appear within the query.

Of all the SQL concepts, GROUP BY has been shown to be one of the most difficult. As with aggregation, GROUP BY is usually called on one (or more) columns within a table. We therefore can apply the same principle here as before, either the attribute can be highlighted, or we can add text. Where Leventidis et al. chose to go with textual representations for aggregation, for GROUP BY they go with a grey highlight on the attribute [25]. In SAVI, Cembalo et al. also chose to display GROUP BY as an highlight on the column they group by [9]. Thalheim mentions that operations, such as GROUP BY, are added to the units, but do not show an example [38].

Besides GROUP BY, JOIN is one of the concepts that students make the most mistakes with. Joining tables is a very abstract concept, as it creates relations between tables that are only implicitly available in the data. A visual representation of this concept can be beneficial, as the explicit drawing of lines, or edges, shows in detail what is implicit in the JOIN condition. These edges also present us with a medium for displaying comparisons between attributes.

One of the most traditional forms of graphs to represent data were Conceptual Graphs [34], which were used to represent database schemas. Conceptual Graphs can be used as graph-based knowledge representations, translating text into graphs, which is similar to our application of SQLVis as a graph translation of the SQL query.

Another system that is similar to SQLVis is QueryVis. QueryVis is a system which simplifies the interpretation of SQL queries by generating visual representations [25]. Their representations are also graph-based, in the sense that each table in the query becomes a separate entity, and there are lines connecting them. The focus in their work is on query interpretation (of existing queries), rather than on query composition. Our study focuses on query formulation, and the use of visual representations to support this.

As discussed in the Introduction, various Visual Querying Systems exist. These VQSs allow users to avoid writing queries textually, instead using visual representations or highlighting data. They thus do not focus on representing the queries themselves. Besides VQS, some Visual Querying Languages exist. One example of a visual querying *language* is QBE [43] which allows for querying databases through visual tables. SQLVis has similarities to QBE in the visual sense, but there are two significant differences. First, our representation is styled as a graph, highlighting the implicit connections between tables in a query by drawing them. Second, our representation is intended as an educational support mechanism and thus requires the users to write their own SQL queries instead of generating the query from the visual representation.

C. Graph visualization principles.

As SQLVis is a graph-based VQR, the design is informed by research on graph visualization.

The leading theory behind drawing static graphs are the Gestalt principles, which represent basic elements of perception, e.g., elements with common features (such as color) are often perceived as groups. Bennett et al. [3] distinguished between principles for combination (similarity, continuation, proximity) and principles for segregation (symmetry, orientation). This use of rules guiding perception can aid in the drawing of graphs as they present good aesthetics [3]. Ware et al. [41] also focus on Gestalt principles in graph drawing. They promote the principle of good continuation: paths in graphs are better perceived if the nodes along the path form a smooth continuous sequence and, furthermore, paths can be emphasized if the line connecting them is curved [41].

In addition to Gestalt principles, graph visualization heuristics have been proposed. These can be divided into categories for node placement, edge placement and graph layout; furthermore, domain-specific heuristics can be applied [3]. Examples of node placement include an even distribution of nodes and the clustering of related nodes, as well as the maximization of node orthogonality [3]. Spatial alignment within a graph is important as it is one of the ways in which cognitive load can be reduced [26]. Edge placement recommends that edges should cross as little as possible, and maximum edge length should be minimized while minimum edge length should be maximized [3]. In addition, edge bends should be uniform. There are various types of graph layout algorithms, including force-directed layouts, attribute-based layouts and constraint-based layouts [18]. A subset of constraint-based layouts are the hierarchical layouts, of which Sugiyama's is the most well-known. A hierarchical layout fits SQL queries well, as there is a hierarchy through the presence of subqueries. Additionally, consistent layout is important because it can preserve the mental map of the viewer [32]. This means that the location of a node should stay the same throughout subsequent graphs, supporting identification as the same node.

Interaction can also be a way to increase graph understanding. One idea is that exploration may reveal insights that were hidden in a static image [40]. Beck et al. [2] argue that interaction with a graph can compensate for worse graph design concerning visualization criteria. In the end, users are essential to the visualization process; Tory et al. [40] therefore argue for the application of human factors and a user-centered design philosophy to be adopted in visualization design.

III. Design

Building on the research literature, we designed SQLVis. In this section we explain all aspects of SQLVis in light of the research presented in Section II-B and Section II-C. Note that SQLVis is a research prototype and therefore incomplete. SQLVis handles all queries that students typically pose in their homework, but lacks functionality for more complex actions, such as view expansion, indexes and insert and delete queries.

Firstly, SQLVis is graph-based and explicitly shows the relations between the tables that are only implicit in the

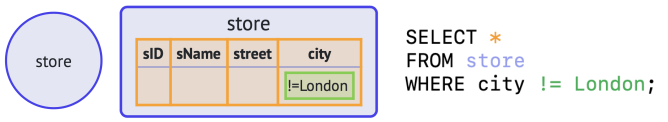


Fig. 3: SQLVis representation for a simple query (collapsed on the left, expanded in the middle).

SQL query itself. All tables are represented as nodes, and all relations are represented as edges. The nodes include the name of the table they refer to, plus an alias in bold if that was defined by the user (see Figure 1 and Figure 4b). SQLVis also allows for interaction with the visualization to increase exploration, as suggested by Tory et al. [40] and Beck et al. [2]. Users of SQLVis can move the representation around, and expand all of the nodes to show the table's schema. Our decision to display the schema horizontally, instead of vertically as other authors have done, gives us more space to add additional content in the tables.

SQLVis highlights any interactions of the query with the table in green and orange (see Figure 3). This calls attention to all elements from the SELECT and WHERE clause, including any comparison values that are present in the query. These highlights for SELECT are different from the approach by Leventidis et al. in QueryVis [25], who chose to create a separate view that includes everything in the SELECT clause. SQLVis does not include such a view, as we do not focus on the result table of the query, but rather on query formulation.

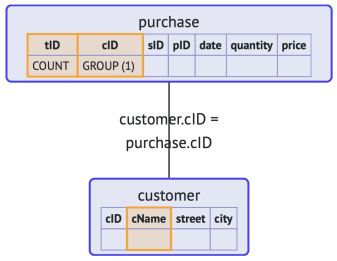
All complex SQL queries contain subqueries and other types of nesting. To visualize these subqueries in the most intuitive way, SQLVis draws boxes around these subqueries as suggested by Thalheim [38]. To distinguish between different subqueries on the same level, and nested subqueries on different levels, SQLVis draws each level of nesting in a different saturation (see Figure 4b). The use of these different colors helps to give an immediate overview of the level of nesting in the query. In case a subquery is negated, for example by using NOT EXISTS or NOT IN, SQLVis displays this negation in words. Other options, such as a different background color or a border for the box led to a very cluttered representation.

Negation may also occur on conditions in the WHERE clause. In that case, SQLVis takes the != or <> from the query and display it in the schema with the text or numbers of the condition, such as !=London in Figure 3.

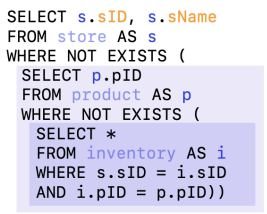
Other elements that cause problems in query formulation are aggregation and GROUP BY. For these two types of actions, SQLVis highlights them in the column that they occur on. This approach is similar to the suggestions by Thalheim [38] and Leventidis et al. [25]. As there may be more than one column used for grouping, SQLVis also includes a number that indicates the grouping order. For an example, see Figure 4a.

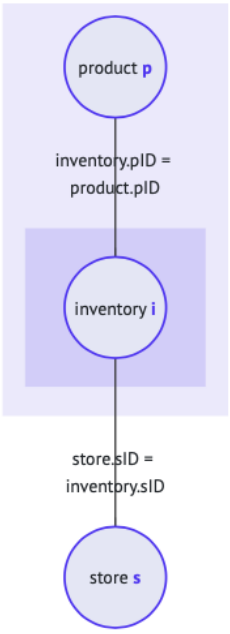
As mentioned before, SQLVis applies a graph-based design to explicitly show any implicit relations. This holds specifically for JOIN conditions, both those using the ON keyword, and the implicit JOIN conditions in the WHERE clause. If the JOIN keyword is used in the query, the JOIN type is also displayed on the edge that contains the JOIN condition.

For the graph visualization principles, we apply as many



(a) A query to count the number of purchases per customer





(b) A query to find stores with all items in stock.

Fig. 4: Two representations of more elaborate queries, containing groupby, aggregation and nested subqueries.

Gestalt principles of grouping in SQLVis as practically possible. After all, each graph represents a single query and thus should be seen as a single entity. Nodes are drawn close together and look the same in shape, size and color. The SQLVis representation is drawn as symmetrical as possible, with subqueries of the same depth drawn on the same level in the graph. This also implies spatial alignment to reduce cognitive load, as suggested by Mayer et al. [26].

Finally, we reflect on the concept of consistent layout as explored by Purchase [32]. This is not applicable in SQLVis, as there is no need to identify nodes: they are identified automatically because they have their name on the face of the node. Furthermore, subsequent questions consider different tables in the database. SQLVis is consistent in the sense that the same query leads to the same layout, it depends on the order in which the tables are called in the query.

In light of the ontology of visual languages by Erwig et al., SQLVis has the following characteristics typical of visual languages: Graph[UnDirected, Labeled] syntactic appearance, 2D and text syntactic features, and [Dynamic] semantics [15].

Note that SQLVis does not support syntax errors: in that case no AST can be parsed, which is required for building the VQR. While visually highlighting syntax errors is straightforward, the independent research question of how Visual Representations can best assist students when syntax errors occur is an interesting challenge outside of the scope of this paper.

IV. Qualitative Study - Methodology

With this first study, we aim to answer Research Question **Q1**: How do users assess SQLVis? To gather feedback on the potential value of the system, we ran a user study in

	Questions
1	List all the product IDs of products that were bought by at most two different customers.
2	List the customers (ID and name) who purchased on the same date both a product with name 'Onions' and a product with the name 'Coffee' (not necessarily at the same store).
3	List the IDs of customers that made a purchase at every store.
4	Find the name of the product that is sold for the highest price (ever) and the name of the store that sold it for that price.

TABLE I: The query formulation problems for the participants of the qualitative study.

Table name	Attributes
customer	cID, cName, street, city
store	sID, sName, street, city
product	pID, pName, suffix
shoppinglist	cID, pID, quantity, date
purchase	tID, cID, sID, pID, date, quantity, price
inventory	sID, pID, date, quantity, unit-price

TABLE II: The database schema, primary keys underlined.

two parts: 20 minutes of query formulation with support of SQLVis and an interview to discuss the design.

Participants. There were five participants with moderate knowledge of SQL. All participants were male with ages between 20 and 28. Their self-reported skill-level was 6 to 8 on a scale from 1-10. The study took 40 minutes and participants were compensated seven euros for participation.

Design. We designed a three-part study. First, we need our participants to experience the system. For usage, we presented them with four query formulation problems, and had them use a system with integrated VQR to solve these problems. We decided not to use a think-aloud method, as writing queries is high in cognitive load, and the talking could disturb our users. Therefore, we had them work undisturbed and discussed the Representation afterwards in a semi-structured interview.

Procedure. At the start of the study, the participants were welcomed and signed an informed consent form. We then presented them with the workings of the query formulation system and answered any questions they might have. They sat down with a laptop to work on several query formulation problems for approximately 25 minutes. See Table I for the questions and the corresponding database schema in Table II. Their interactions with the system were screen-recorded. After getting to know the system, we finished with interview questions about the system, and any other questions or discussion points the participants may have had.

V. Qualitative study - Results

In this section we focus on the semi-structured interview. In the interview, we asked the participants what their general impression of the VQR was. Below we discuss some quotes from our participants.

> "[SQLVis] is a good idea. It is more organized than looking at tables, you can see exactly the parts you need instead of having to search through large tables." - participant 1

Our first participant is happy with the structure that SQLVis brings. One advantage of the representation is that the nodes can be expanded to show their internal schema. This can help the user find the appropriate attributes to use within their query.

> "The design makes sense, for example using highlights to show which columns you are selecting." - participant 2

The highlights are appreciated by our second participant. Besides the node-link structure depicting the query's implicit relations, the highlights look at the specifics of the selections and conditions. This helps to find any mistakes in the query (for example using an incorrect table), and help the user to finalize the query.

Participant 3 refers to the use of the VQR as a tool to keep track of the current state of the query: has the user incorporated all the elements they wanted and thus completed their query?

> "I think [SQLVis] is a good idea, because you can often lose track of what exactly you are [writing], what columns you have selected etcetera. It helps with that." - participant 3

During query formulation, users need to keep track of many aspects such as the database schema and which parts of the question they have already translated. This causes a high cognitive load. SQLVis shows all operators and operands of the query in its current state, thereby lowering the cognitive load associated with query formulation.

Participant 4 looks at this from another direction:

> "SQLVis was fun to see but I have not used it a lot. Mostly because the visualization only shows what you already have, which does not help you move forward." - participant 4

Indeed, SQLVis will not assist the user by suggesting elements to include. As we aim to support learning, we encourage learners to formulate queries themselves rather than automating part of the query formulation process.

Participant 4 was able to finish all four query formulation problems. As the queries were easy for them, there was no need to use SQLVis as a support mechanism. For struggling students however, the value can be quite high.

> "The VQR looks good but I didn't really use it. Mostly because I am not used to having such functionality. If I were to become more familiar with the system I think it could be helpful to find mistakes in your queries. Finding mistakes is difficult for complex queries." - participant 5

Participant 5 highlighted the fact that as SQLVis was still unfamiliar to them, they did not use it a lot. However, from the introduction of it at the start of the study, they suggested that it would be useful for finding mistakes in the query.

Summary. Overall our participants were positive about the application of SQLVis as a support mechanism for query formulation. The design was described as sensible, clear and fun. Possible applications mentioned by participants include finding mistakes and keeping track of query formulation. The qualitative study, however, was too short for participants to get used to incorporating it into the query formulation process.

	Questions
query3_2	Select all distinct combinations of names and ids of customers who have both a shoppinglist and a purchase, both for the same date in 2018.
query3_3	Select all different names and ids of the customers who never made a purchase at a store with the name 'Kumar'. Also include the customers who never made a purchase, but are still represented in the customer relation of our database.
query3_4	Select all different names and ids of customers who made at least one purchase at a store with the name 'Kumar' and never a purchase at a store with a different name than 'Kumar'.
query4_3	Write a SQL query for the following RA query: $\Pi_{sName,city}(store) \div (\Pi_{city}(customer) \bigcup \Pi_{city}(store))$
query4_4	List the names of all of those customers, and only of those customers, that spent an amount of money on any one date that is at least 75% of the maximal amount of money ever spent by a single customer on a single day.
query4_5	Per city, how many customers living in the city have made at least one purchase from a store in London?

TABLE III: The questions for the quantitative study. The first number in the question name refers to the week they were posed.

VI. Quantitative study - Methodology

In this second study, we aim to answer Research Question **Q2**: What is the effect of using SQLVis on the query formulation process? To answer this question we asked Bachelor students taking our second-year databases course to write SQL queries through Jupyter Notebooks. We divided the students in two groups (a between-subjects protocol), using two different notebooks: one plain (control group), and one that included SQLVis as a Python package. Our participants were divided over the groups by last name. This random division over the groups should make sure that the groups are balanced in terms of skills and demographics.

Design. To compare performance in an AB-test, we needed to find a way to have people work on the same problems in both conditions. The opportunity arose to use the authors' institution's second year introductory Databases course. To analyze not only the final results, but all intermediate attempts, we set up a Jupyter notebook. Students at the authors' institution are familiar with writing SQL queries through Jupyter notebooks at this point of the Bachelor program. This, in combination with the flexibility that Python offers, is the reason we went with notebooks. We created two similar notebooks, both including the homework questions, with one version including SQLVis. To invoke SQLVis, students can call it's visualize() method, which uses the database schema and query text to generate a visual representation and displays this directly below the code cell that contains the visualize() call. Additionally, the notebooks logged all interactions of the student, including timestamps.

Materials. Notebooks were offered for two different homework assignments in weeks 3 and 4 of the course. The notebooks contained cells with a description of how to participate, the problem formulation for each of the problems, and room to formulate answers. Students could use the notebooks locally. Usage of the notebooks generated Python logs in separate files. For the non-visual notebook, there was just one cell available for each question, that allowed the participants to define and execute a query at the same time. For the visual notebooks (including SQLVis), there were three predefined types of cells: define, visualize and execute. Execution of each of these cells is logged as an action of correspondingly named type.

The query formulation problems included in the notebooks are listed in Table III. This study used the same database schema (Table II) as the qualitative study described above.

Procedure. Students were offered the study's notebook through the University's Learning Management System (LMS). They were informed through the LMS that the notebooks were uploaded, and received instructions on which variant of the notebooks to use and how to submit. Upon submission of their homework in the LMS, students were asked to also submit any logs they may have generated. These logs were then downloaded and organized.

	Non-visual	Visual
Participant count	27	16
Total interactions	3377	2624
Total executed	3377	921
Count correct executed	174	111
Count incorrect executed	2216	496
Count error executed	960	310

TABLE IV: Statistics on query execution by our participants.

VII. Quantitative study - Results

In this section we call the notebooks that include SQLVis 'visual', and the control condition 'non-visual'. Answers can be either correct (the result table of the attempt corresponds to a pre-defined correct result table), incorrect (the result tables do not correspond), or erroneous (the DBMS returns an error upon execution of the query).

First, we'll present general numbers on participation in Table IV. The table shows that the groups were not equal in size. However, the number of interactions with the notebooks shows that the size of the data is adequate for analysis. As we are evaluating performance in this study, we look at the four different outcomes that a query could have: correct, incorrect, producing an error, or resulting in a time-out. For both groups, we see that incorrect answers are most common (65.6 and 53.9 percent), followed by errors (28.4 and 33.6 percent).

Correctness and attempts. To see how these correct answers are distributed, we analyze per query and per participant whether they found at least one correct answer for each question. The results can be found in Figure 5. The proportion of participants with a correct answer for the first question is approximately equal. Then for query3_3 until query4_3, the participants in the visual condition score better. It is interesting to see that the majority of students were not able to find a correct answer for query4_4 and query4_5. If we compare this to the number of attempts made on each question, as shown in Table V, we see a difference in behavior between the non-visual and visual notebooks. The participants with visual notebooks have more attempts on the first four questions (given that they have less attempts overall), and much less

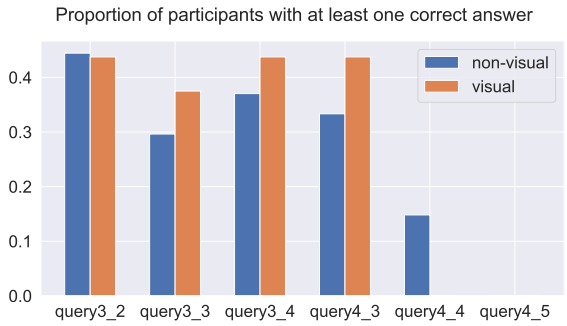


Fig. 5: Counting the proportion of participants having at least one correct answer per question.

attempts than the non-visual participants for the last two questions. For the visual notebooks, there is an outlier on query4_3 specifically, where a lot of attempts were made. This is not due to any one participant, but an effect of increase over multiple participants. In the other direction, query3_4 is an outlier with low number of attempts in both conditions. This may be due to the fact that the question is closely related to the question before it. Participants could reuse parts of their query to answer the question more quickly.

Statistical testing. We hypothesize that students using the visual notebooks are more effective query formulators. This means: more correct answers, fewer incorrect answers, and fewer errors. Because of the differing group sizes, we test this via proportion Z-tests. For the visual notebooks, we just include the 'execute' cells, as there is a duplicate-query effect for define cells (which are counted as correct or incorrect too).

For correct answers, we have a proportion of 12.1 percent for visual notebooks and a proportion of 5 percent for non-visual notebooks. The null hypothesis asserts equal means, but the calculation of a one-sided proportion z-test shows that the proportion of correct answers for the visual notebooks is significantly higher than that of the non-visual notebooks, with a p-value of 4.35e-14. For incorrect answers, we have a proportion of 53.8 percent for visual notebooks and a proportion of 65.6 percent for non-visual notebooks. The null hypothesis asserts equal means, but the calculation of a one-sided proportion z-test shows that the proportion of incorrect answers for the visual notebooks is significantly lower than that of the non-visual notebooks, with a p-value of 2.7e-11. For error-producing answers, we have a proportion of 33.6 percent for visual notebooks and a proportion of 28.5 percent for non-visual notebooks. The null hypothesis asserts equal means, but the calculation of a one-sided proportion z-test shows that the proportion of erroneous answers for the visual notebooks is significantly higher than for the non-visual notebooks, with a p-value of 0.001. The absence of advantage for the visual condition can be explained by the fact that the representation cannot be generated when the query contains a syntax error, as parsing the query's syntax tree fails in that case.

Higher efficiency can also be expressed in a lower number of attempts. We compare the means via a t-test (again taking only the 'execute' interactions). The null hypothesis asserts equal means. We take an alternative hypothesis that asserts lower means for visual notebooks. We find that we can reject the null hypothesis with a p-value of 0.01.

Query	q3_2	q3_3	q3_4	q4_3	q4_4	q4_5
Non-visual	460	587	299	686	696	649
Visual	149	142	101	242	147	140

TABLE V: Statistics on query execution by our participants.

Finally, we can express efficiency in time taken to solve the questions. We found that participants in the visual condition took 2 hours and 4 minutes on average, participants in the non-visual condition on average took 2 hours and 25 minutes to finish the exercises. A t-test did not show a significant difference between the conditions.

Error analysis. In short, participants in the visual notebooks had fewer attempts, more correct answers, fewer incorrect answers, and more errors, than participants without the visual representation. To gain more insights into these errors, we undertook text processing on the errors returned by the database system, to find out what the most common types of errors were. In summary, the most common error is the plain syntax error, which represents 58.5 percent of errors in non-visual notebooks, and 33.5 percent of errors for visual notebooks. Other noteworthy errors that are made more in the non-visual notebooks are *incomplete output*: 3.75 versus 1.6 percent, and errors regarding the number of returned columns not matching the number of expected columns: 7 vs 4.5 percent. For the visual notebooks, an error that occurs more often is *ambiguous column name*: 5 percent for non-visual notebooks vs 13.2 percent for visual notebooks. Participants in both conditions have similar counts for *no such column*: 15.4 versus 16.4 occurrences.

Although the visual representation did not work in cases of syntax errors, we can draw a parallel between the error types and the visual representation usage, as query definition is a process of refinement. For example, participants using the visual representation might have a hard time with correct column names as the visual representation extracts the correct ones from the schema automatically. From the lower incidence of general syntax errors, it seems that the participants in the visual condition were more careful in their query formulation and thus made more meaningful errors.

SQLVis usage. We define three different interactions in the visual notebook: define, visualize and execute. Usage statistics show that define was called 1037 times, execute was called 921 times and visualize was called 769 times. As the number of define calls is higher than the number of execute calls, it seems that the visualization helped in the refinement of query formulation. The fact that using the visual representation was optional, and our 16 participants chose to use it 769 times is another indication of usefulness.

To analyze SQLVis quantitatively, we define helper patterns:

1) Execute a query, visualize this query, define a new query.
2) Visualize a query, define a new query, visualize this query.
3) Execute a query, define a new query, visualize this query.
4) Define a query, visualize this query, define a new query.

These patterns highlight the cases in which the visual representation is used as a tool for repair, verification or query building. For all patterns, we require that they are subsequent attempts

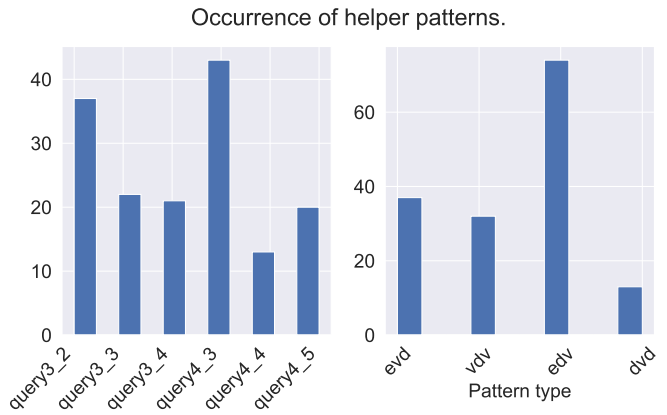


Fig. 6: The division of helper patterns over the homework questions. The pattern types are identified by the first letter of each action, so evd represents execute, visualize, define.

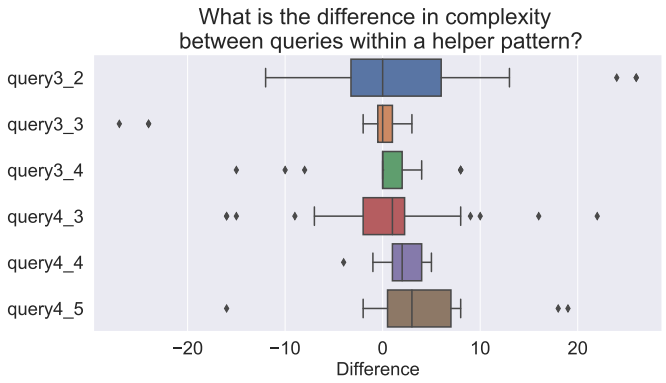


Fig. 7: The usage of helper patterns leads to new queries. What is the difference in Bowen complexity score between the old and the new query?

on the same question, the first found query was incorrect, and they cannot include syntax errors as in those cases the visual representation can't be used for problem-solving purposes.

We first count the incidence of these helper patterns together: they occur 156 times in total. In Figure 6, they are divided over their corresponding question and counted by type. Helper pattern 1 was used most on query4_3 and query4_5, helper pattern 3 was used most on query4_3, and helper patterns 2 and 4 were used most on query3_2. Overall, the highest incidence of repair patterns occurs for query3_2 and query4_3. As query4_3 requires a translation from the Relational Algebra to SQL, the addition of an intermediate representation might have been especially valuable for the participants. For the pattern types themselves, we see that pattern 3 is the most popular. This may be explained by the order provided in the notebook: define, visualize, execute, of which pattern 3 is a variant.

Diving deeper into these patterns, we investigated duration. A pattern was only defined as such if it was shorter than fifteen minutes, although we found that the maximum duration of these patterns was close to six minutes. The average duration was one minute and two seconds. We found a minimum duration of zero seconds, which must result from the *Run all cells* functionality in Jupyter. Therefore, we decided to exclude all patterns of less than ten seconds. This led to a new mean duration of one minute and fifteen seconds.

Another aspect of these helper patterns is the change in query complexity. We define complexity as the number of operators and operands in the query, an idea posed by Bowen et al. [6]. We took each instance of the helper patterns and calculated the difference in query complexity between the first and the last query in the pattern of three. The scores where then categorized by question and plotted in a box plot, see Figure 7. We see that for the questions in week 3, the median complexity difference is 0. For week 4, medians are above 0, with an average scores of 0.5 to 3.5. This means that on average, at least one operator or operand was added to the query. It seems that in week 4, the helper patterns on averaged increased complexity of the queries, which means participants were using SQLVis as query formulation support.

Summary. Our participants actively used SQLVis, with 769 calls to the visualize command. Participants using SQLVis wrote significantly more correct queries than the control group, and significantly fewer incorrect queries. SQLVis also facilitated efficiency, with a significantly lower number of attempts. Our definition of helper patterns showed that our participants used SQLVis as a tool for query formulation and verification.

VIII. Conclusion and Discussion

In this paper, we presented SQLVis, a Visual Query Representation for supporting query formulation. We evaluated SQLVis through both a qualitative and quantitative study. We posed Research Question **Q1**: How do users assess SQLVis? We found that our participants were positive about the design, but had a difficult time applying it as the study was too short to become familiar with SQLVis. We followed up by posing Research Question **Q2**: What is the effect of using SQLVis on the query formulation process? We found that our participants actively used the visualization (769 calls) and in week 4 they used SQLVis to build, repair and verify queries. This increased usage in week 4 could be a reflection of the qualitative study, where it took our users time to learn to take full advantage of SQLVis. In addition, SQLVis users performed better in terms of correctness when compared to the control group.

SQLVis has been built as a research prototype to analyze the applicability and benefits of VQR systems for SQL education. As we have found evidence that SQLVis has a positive effect on the query formulation process, it provides a foundation for future research. We close with three pointers for future work. First, we highlighted in Section III that visual representation of syntax errors to support learners is an important challenge for further study. Second, our research was conducted at only one institution, with students taking a single course. To diminish the effect of the single lecturer, it would be interesting to run our study in additional institutions. Finally, we have shown the effect of using SQLVis on correctness and efficiency through number of attempts. From prior literature we know this may be due to parallel processing allowed by showing a visual representation. It would be interesting to continue this work by verifying the effect of SQLVis on cognitive load.

REFERENCES

[1] Alireza Ahadi, Julia Prior, Vahid Behbood, and Raymond Lister. A Quantitative Study of the Relative Difficulty for Novices of Writing Seven Different Types of SQL Queries. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, pages 201–206, 2015.

[2] Fabian Beck, Michael Burch, and Stephan Diehl. Towards an aesthetic dimensions framework for dynamic graph visualisations. *Proceedings of the International Conference on Information Visualisation*, pages 592–597, 2009.

[3] Chris Bennett, Jody Ryall, Leo Spalteholz, and Amy Gooch. The aesthetics of graph visualization. *Computational Aesthetics in Graphics, Visualization, and Imaging*, pages 1–8, 2007.

[4] Sourav S Bhowmick. DB x HCI: towards bridging the chasm between graph data management and HCI. In *Database and Expert Systems Applications*, pages 1–11. Springer, 2014.

[5] Harold Boley. Grailog 1.0: Graph-Logic Visualization of Ontologies and Rules. In *International Workshop on Rules and Rule Markup Languages for the Semantic Web.*, 2013.

[6] Paul L. Bowen, Colin B. Ferguson, Timothy H. Lehmann, and Fiona H. Rohde. Cognitive style factors affecting database query performance. *International Journal of Accounting Information Systems*, 4:251–273, 2003.

[7] Stefan Brass and Christian Goldberg. Semantic errors in SQL queries: A quite complete list. *Journal of Systems and Software*, 79(5):630–644, 2006.

[8] Tiziana Catarci, Massimo Mecella, Stephen Kimani, and Giuseppe Santucci. Visual Query Interfaces. In *The Wiley Handbook of Human Computer Interaction*, pages 561–577. John Wiley & Sons, Ltd, Chichester, UK, dec 2017.

[9] Maurizio Cembalo, Alfredo De Santis, and Ferraro Petrillo Umberto. SAVI: A new system for advanced SQL visualization. *SIGITE'11 - Proceedings of the 2011 ACM Special Interest Group for Information Technology Education Conference*, pages 165–170, 2011.

[10] Hock Chuan Chan. A Two-stage Evaluation of User Query Performance for the Relational Model and SQL. In *PACIS 2007 Proceedings*, pages 213–227, 2007.

[11] Hock Chuan Chan, Bernard C. Y. Tan, and Kwok Kee Wei. Three important determinants of user performance for database retrieval. *International Journal of Human-Computer Studies*, 51(5):895–918, nov 1999.

[12] Michel Chein and Marie-Laure Mugnier. Conceptual Graphs with Negation. In Lakhmi Jain and Xindong Wu, editors, *Graph-based Knowledge Representation*, chapter 12, pages 337–377. Springer-Verlag London, 2009.

[13] Michelle Patrick Cook. Visual Representations in Science Education: The Influence of Prior Knowledge and Cognitive Load Theory on Instructional Design Principles. *Science Education*, 90(6):1073–1091, 2007.

[14] Michael De Raadt, Stijn Dekeyser, and Tien Yu Lee. Do students SQLify? Improving learning outcomes with peer review and enhanced computer assisted assessment of querying skills. *ACM International Conference Proceeding Series*, 276:101–108, 2006.

[15] Martin Erwig, Karl Smeltzer, and Xiangyu Wang. What is a visual language? *Journal of Visual Languages and Computing*, 38(October 2016):9–17, 2017.

[16] Sneha Gathani, Peter Lim, and Leilani Battle. Debugging database queries: A survey of tools, techniques, and users. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, page 1–16, 2020.

[17] Wolfgang Gatterbauer. Databases will Visualize Queries too. *Proceedings of the VLDB Endowment*, 4(12), 2011.

[18] Helen Gibson, Joe Faith, and Paul Vickers. A survey of two-dimensional graph layout techniques for information visualisation. *Information Visualization*, 12(3-4):324–357, 2013.

[19] Weidong Huang, Peter Eades, and Seok Hee Hong. Measuring effectiveness of graph visualizations: A cognitive load perspective. *Information Visualization*, 8(3):139–152, 2009.

[20] Weidong Huang, Seok-Hee Hong, and Peter Eades. Predicting graph reading performance: a cognitive approach. In *Asia-Pacific Symposium on Information Visualisation*, pages 207–216, 2006.

[21] Hv Jagadish, Adriane Chapman, Aaron Elkiss, Magesh Jayapandian, Yunyao Li, Arnab Nandi, and Cong Yu. Making database systems usable. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, pages 13–24, 2007.

[22] Changjiu Jin, Sourav S Bhowmick, Byron Choi, and Shuigeng Zhou. Prague: towards blending practical visual subgraph query formulation and query processing. In *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, pages 222–233. IEEE, 2012.

[23] R Kearns, Stephen Shead, and Alan Fekete. A teaching system for sql. In *Proceedings of the 2nd Australasian conference on Computer science education*, pages 224–231, 1997.

[24] Gordon Kindlmann and Carlos Scheidegger. An Algebraic Process for Visualization Design. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2181–2190, dec 2014.

[25] Aristotelis Leventidis, Jiahui Zhang, Cody Dunne, Wolfgang Gatterbauer, HV Jagadish, and Mirek Riedewald. Queryvis: Logic-based diagrams help users understand complicated sql queries faster. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 2303–2318, 2020.

[26] Richard E. Mayer and Roxana Moreno. Nine Ways to Reduce Cognitive Load in Multimedia Learning. *Educational Psychologist*, 38(1):43–52, 2010.

[27] Daphne Miedema. Towards successful interaction between humans and databases. Master's thesis, Eindhoven University of Technology, 2019.

[28] Antonija Mitrovic. Learning SQL with a computerized tutor. In *ACM SIGCSE Bulletin*, pages 307–311, 1998.

[29] Hiroyuki Nagataki, Yoshiaki Nakano, Midori Nobe, Tatsuya Tohyama, and Susumu Kanemune. A visual learning tool for database operation. *ACM International Conference Proceeding Series*, pages 39–40, 2013.

[30] George Obaido, Abejide Ade-Ibijola, and Hima Vadapalli. Generating SQL queries from visual specifications. *Communications in Computer and Information Science*, 963:315–330, 2019.

[31] William C. Ogden and Susan R. Brooks. Query languages for the casual user. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems - CHI '83*, pages 161–165, New York, New York, USA, 1983. ACM Press.

[32] Helen C. Purchase, Eve Hoggan, and Carsten Görg. How important is the "mental map"? - An empirical investigation of a dynamic graph layout algorithm. *Lecture Notes in Computer Science*, 4372:184–195, 2007.

[33] Wolfgang Schnotz and Maria Bannert. Construction and interference in learning from multiple representation. *Learning and Instruction*, 13(2):141–156, 2003.

[34] John F Sowa. Conceptual graphs. *Foundations of Artificial Intelligence*, 3:213–237, 2008.

[35] Ahmet Soylu, Martin Giese, Ernesto Jimenez-Ruiz, Guillermo Vega-Gorgojo, and Ian Horrocks. Experiencing OptiqueVQS: a multi-paradigm and ontology-based visual query system for end users. *Universal Access in the Information Society*, 15(1):129–152, 2016.

[36] John Sweller. Cognitive load during problem solving: Effects on learning. *Cognitive Science*, 12(2):257–285, 1988.

[37] Toni Taipalus and Mikko Siponen. Errors and Complications in SQL Query Formulation. *ACM Transactions on Computing Education*, 18(3), 2018.

[38] Bernhard Thalheim. Visual SQL- high-quality ER-Based query treatment. *Lecture Notes in Computer Science*, 2003.

[39] Bernhard Thalheim. Visual sql: towards er-based object-relational database querying. In *International Conference on Conceptual Modeling*, pages 520–521. Springer, 2008.

[40] Melanie Tory and Torsten Moller. Human factors in visualization research. *IEEE transactions on visualization and computer graphics*, 10(1):72–84, 2004.

[41] Colin Ware, Helen Purchase, Linda Colpoys, and Matthew McGill. Cognitive measurements of graph aesthetics. *Information Visualization*, 1(2):103–110, 2002.

[42] Hsin I. Yung and Fred Paas. Effects of computer-based visual representation on mathematics learning and cognitive load. *Educational Technology and Society*, 18(4):70–77, 2015.

[43] Moshé M. Zloof. Query by Example. In *Proceedings of the May 19-22, 1975, National Computer Conference and Exposition*, pages 431–438, New York, 1975. IBM T.J. Watson Research Center.