Mental Workload Impact of a Visual Language on Understanding SQL Queries

Edson A. Silva, Natália M. Franco, Márcio R. Ferro, Robson N. Fidalgo

Centro de Informática — Universidade Federal de Pernambuco (UFPE)
CEP 50740-560 – Recife – PE – Brasil

{eas4, nmf, mrcf, rdnf}@cin.ufpe.br

Abstract. *Structured Query Language (SQL) is an essential topic in introductory database courses. Despite its declarative syntax, SQL query specification can be confusing. Visual Query Languages (VQLs) are an alternative to reduce this complexity. However, these VQLs do not cover many complex constructions. To overcome this limitation, we present the Diagrammatic SQL (DSQL). Because increased Mental Workload (MW) during task performance may increase fatigue and facilitate errors, our objective is to investigate the MW impact of DSQL on the understanding of SQL queries. In this experiment, we compare the accuracy, time, and ease-of-understanding complex queries using both languages. The results indicate that there is no significant difference.*

1. Introduction

In academics, the most important computer science university curricula [ACM/IEEE 2013, ACM/IEEE 2017] recommend the teaching of Structured Query Language (SQL). In the industry, SQL is the default language for querying relational databases and its use as a query interface for non-relational database (e.g., Spanner or Spark) is increasing. Moreover, in 2018, SQL remains the second most-used language for the second consecutive year and is among the most popular languages for Web developers (2nd place), desktop developers (1st place), system administrators/DevOps (2nd place), and data scientists (2nd place) [StackOverflow 2018]. In other words, SQL is a relevant and widely-used language in both academics and industry.

Although the basic syntax of SQL is easy to understand, specifying queries that use complex constructs is a non-trivial task. That is, whenever a query has at least one of the following constructs, it can be considered complex, and will therefore require greater intellectual and technical effort: 1) subquery (correlated or non-correlated) [Czejdo et al. 1993, Kawash 2004, Zhang et al. 1999, Catarci and Santucci 1995]; 2) join (Cross, Natural, Inner, or Outer) [Catarci and Santucci 1995]; 3) set operation (Union, Except, and Intersect) [El-Mahgary and Soisalon-Soininen 2015]; 4) conditional expression (Case clause) [Gryz et al. 2008]; and 5) grouping restriction (Having clause) [Zhang et al. 1999].

Considering that SQL queries can involve complex constructs, a Visual Query Language (VQL) is an alternative that can reduce this complexity (cf. Section 2.1). Despite the potential advantages of VQLs, in practice, they do not cover most of the complex constructs of SQL. This is confirmed by [Abouzied et al. 2012] and [Bakke and Karger 2016], who state that, although there have been recent studies of VQL, little has been done to improve then since the initial proposals. To mitigate this problem, we present the graphical notation for Diagrammatic Structured Query Language (DSQL) (cf. Section 3), a VQL that aims to be a visual alternative for modeling complex queries in SQL. As a VQL, DSQL promotes the development of creative and critical thinking by providing visual abstractions that can help improve reasoning and problem-solving skills before coding in SQL.

Mental Workload (MW) is characterized as a reflection of mental stress related to a task performed, its environment, and its specific operating conditions, along with the worker's ability to respond to these demands [Cain 2007]. In general, the mental workload associated with an easy task is low, whereas difficult tasks produce a higher mental workload. Increased mental workload during the performance of a task may increase fatigue and facilitate the commission of errors [Cain 2007, Longo 2015]. In this paper, our objective is to investigate the MW impact of DSQL on the understanding of SQL queries. In an experiment, we compare the accuracy, time, and ease-of-understanding of complex queries using both languages. To compare the ease-of-understanding, we used NASA-TLX (cf. Section 2.2) because it is one of the most commonly-used instruments for measuring subjective MW [Cain 2007].

The remainder of this paper is organized as follows: Section 2 introduces the main concepts related to Visual Query Languages and NASA-TLX; Section 3 specifies the DSQL notation; Section 4 shows an overview of the main related proposals; Section 5 discusses our experimental evaluation; and Section 6 presents our final considerations.

2. Background

2.1. Visual Query Language

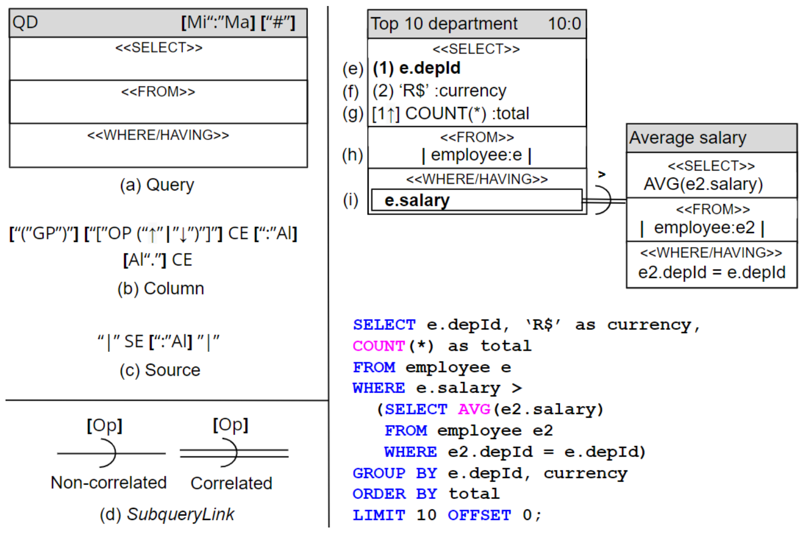
Visual Query Language (VQL) is an alternative that promotes the understanding and critical analysis of complex queries by using visual abstractions. That is, different from SQL textual representation, VQL graphical abstractions help to focus on the semantic aspects of the query rather than on its syntax [Catarci and Santucci 1995, Hvorecký et al. 2010]. It is possible to produce a VQL with only pen and paper, however, a VQL is more powerful when it is implemented by a Computer-Aided Software Engineering (CASE) tool, because these tools facilitate interaction with the VQL notation and can prevent or detect errors and generate software artifacts (e.g., source code or documentation) [Brambilla et al. 2017]. In this context, it is important to highlight that, before developing the CASE tool, the VQL graphical notation must be specified.

2.2. NASA-TLX

The NASA Task Load Index (NASA-TLX) [Hart and Staveland 1988] is a multidimensional evaluation procedure that provides an overall workload score based on a weighted average of evaluations in six dimensions: mental, physical, and temporal task demands; and effort, frustration, and perceived performance. The NASA-TLX is applied two steps: scoring and weighting. In the scoring phase, the participant should assigns a value on a scale of 0 to 100 for the workload in each of the six dimensions. The scale is divided into intervals of five units. In the weighting phase, the participant defines the relevant load factors. The participants are presented with fifteen binary comparisons of the six dimensions and must choose which of each pair he perceives as the largest source of load. This makes it possible to weight each of the factors as a function of the number of times it appears. Each factor can receive a weight from zero (it does not even appear once and therefore is not considered relevant) up to five (the factor was chosen every time and therefore is considered the most important load source). The number of times each factor is selected is counted. The score obtained from the first phase of the method (converted to a scale of 100) is then multiplied by the value obtained from the weighting phase for each dimension. Dividing the sum of all values by 15 (the total number of binary combinations) gives the weighted average of the global workload for the task studied.

3. Diagrammatic Structured Query Language

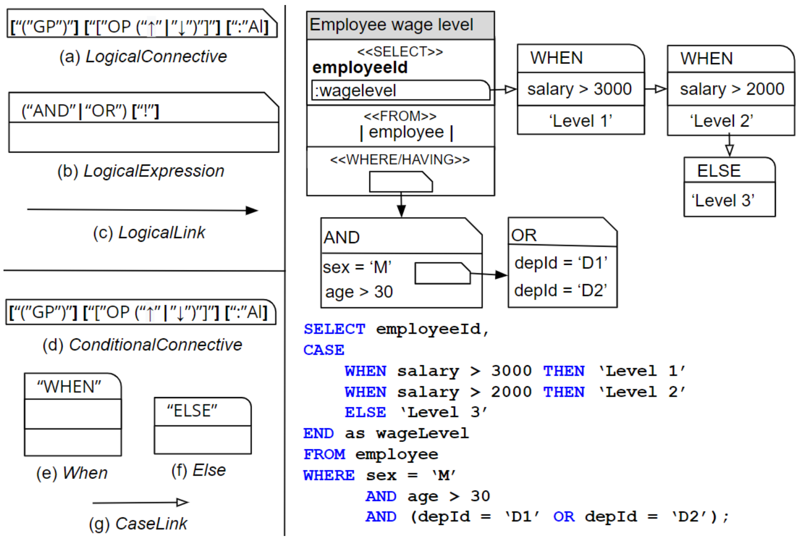
Figures 1, 2, 3, and 4 show the DSQL notation, which is based on the diagrammatic representation of the UML Class diagram, using the following representations: boxes, compartments, and links.



```sql
SELECT e.depId, ‘R$’ as currency,
COUNT(*) as total
FROM employee e
WHERE e.salary >
   (SELECT AVG(e2.salary)
    FROM employee e2
    WHERE e2.depId = e.depId)
GROUP BY e.depId, currency
ORDER BY total
LIMIT 10 OFFSET 0;
```

Figure 1. *Query*, *Column*, *Source*, and *SubqueryLink* constructors.

Figure 1 presents the *Query*, *Column*, *Source*, and *SubqueryLink* constructors of DSQL. The Query constructor (Figure 1-a) can represent either a query or a subquery. This constructor has four compartments. The top compartment has a label formed by the query description, its minimum and maximum result limits ([Min“:”Max]), and whether or not it is distinct ([“#”]). The other three compartments, from top to bottom, map the following concepts, respectively: Select, From, and Where/Having. The Column constructor (Figure 1-b) can represent a field (boldface – Figure 1-e), a free expression (regular font – Figures 1-f and 1-g), a conditional expression (Figure 2), or a subquery (Figure 1-i). When Column is used in the Select compartment, a group position ([“(”GP“)”]), an order position (OP), and a sort type (ST - [“[”OP ST“]”]) can be set. The sort type can be ascending = ↑ or descending = ↓. We highlight the following restrictions: (1) The column order in the Select compartment (from top to bottom), the number in the parentheses, and the number in the brackets define the positions of Select, Group By, and Order By, respectively; (2) Free expressions must follow the rules of the DBMS and can be used to write any SQL function, operation, or literal; and (3) when a free expression with an aggregate function is specified in the Where/Having compartment, this grouping restriction will be mapped to a condition in the Having clause. The Source constructor (Figure 1-c) can be used to represent a table (Figure 1-h) or a subquery. When a source element (SE) is a table, its label must describe the name of the table, otherwise, its label will be empty. In both cases, an alias (Al) is optional (“|”SE[“:”Al]“|”). The SubqueryLink (Figure 1-d) constructor connects the Column or Source constructors to a subquery, showing the subset/inclusion relations. When a SubqueryLink is connected to a Column, an arithmetic or comparative operation can be specified in the label. The right side of Figure 1 shows a query using these DSQL constructors and its representation in SQL. This query answers the following question: *"What are the top ten departments with the most employees who receive more than the average department salary, ordered by total number of employees?"*.

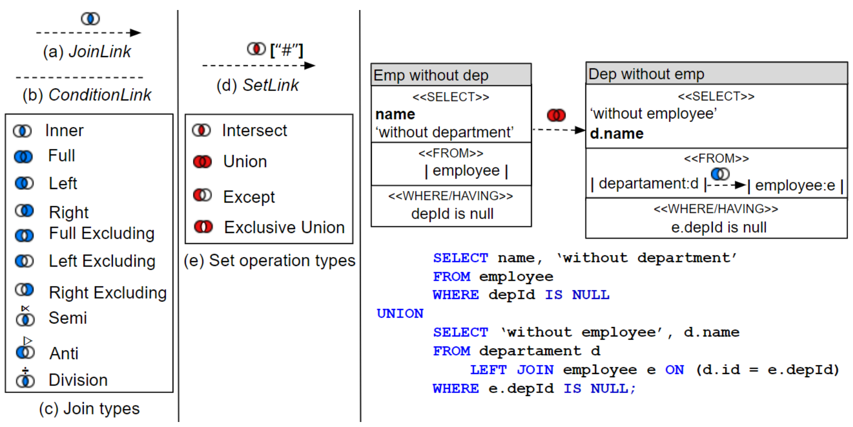


```
SELECT employeeId,
CASE
     WHEN salary > 3000 THEN 'Level 1'
     WHEN salary > 2000 THEN 'Level 2'
     ELSE 'Level 3'
END as wageLevel
FROM employee
WHERE sex = 'M'
        AND age > 30
        AND (depId = 'D1' OR depId = 'D2');
```

Figure 2. Constructors for logical connectors and conditional expressions.

Figure 2 presents the logical connectors and conditional expression constructors of DSQL. When a logical expression has many operations, a visual resource that makes understanding easier is the modularization of its operations using logical connectives. In this sense, three constructors are required: (1) LogicalConnective (Figure 2-a), a constructor that indicates a logical connective to be used; (2) LogicalExpression (Figure 2-b), a constructor that gathers the set of conjunction or disjunction operations; and (3) LogicalLink (Figure 2-c), a constructor that shows the execution sequence of the logical connectives. The LogicalConnective constructor has only one compartment, containing a label. Being a Column constructor, this label contains the group position, order position, and sort type. However, the label does not have the column element properties. The LogicalExpression constructor has two compartments. The upper one specifies whether the logical expression is a set of conjunctions or disjunctions, as well as their negations (("AND" | "OR")["!"]). The lower compartment gathers the set of logical operations. Because a conditional expression can also have many operations, its graphical notation is also based on boxes, compartments, and links to modularize its operations. Four constructors are required to specify conditional expressions: (1) ConditionalConnective (Figure 2-d), whose label is equal to *LogicalConnective*; (2) When (Figure 2-e), which has three compartments: the first corresponding to its name ("WHEN"), the second used to express a logical expression to be evaluated, and the third to specify a DSQL constructor (e.g., field, subquery, free expression, or another conditional expression) to be executed when the logical expression is true; (3) Else (Figure 2-f), which has two compartments: the first corresponding to its name ("ELSE") and the second used to specify a DSQL constructor that will be executed if no When logical expressions are true; and (4) CaseLink (Figure 2-g), which shows the CASE execution sequence. The right side of Figure 2 illustrates a DSQL query that uses these constructors and shows their respective representations in

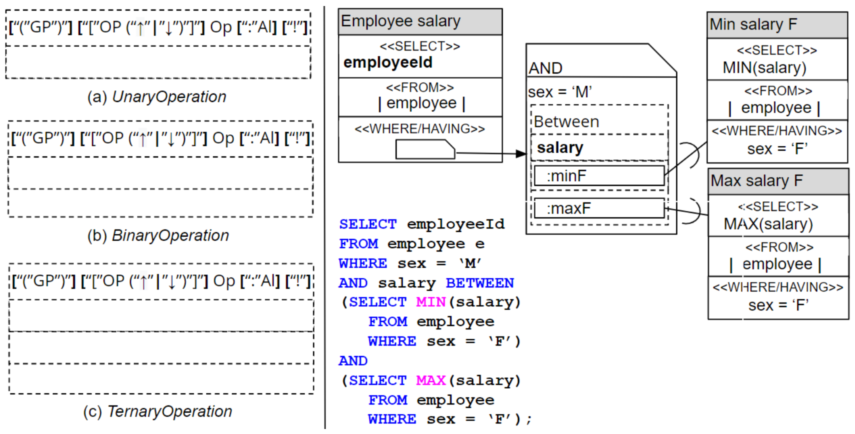
SQL. This query answers the following question: *"What would the wage level of male employees over 30 who work in department D1 or D2 be, where Levels 1, 2, and 3 had wage ranges of: greater than 3000, greater than 2000, and less than or equal to 2000, respectively?"*.



```
      SELECT name, 'without department'
      FROM employee
      WHERE depId IS NULL
UNION
      SELECT 'without employee', d.name
      FROM departament d
          LEFT JOIN employee e ON (d.id = e.depId)
      WHERE e.depId IS NULL;
```

Figure 3. Constructors for join and set operations.

Figure 3 presents the join and set operations constructors of DSQL. A join is specified with the JoinLink constructor (Figure 3-a), which can abstract a join (i.e., Inner join, Left outer join, Right outer join, or Full outer join), a manipulation of a join (i.e., Left excluding join, Right excluding join, or Full excluding join), or a pseudo-join (i.e., Semi join, Anti join, or Division) (Figure 3-c). When it is necessary to specify a Theta join or Equi join whose condition is not based on primary or foreign keys, the ConditionLink (Figure 3-b) constructor must be used to specify these particular conditions. Set operations are specified with the SetLink (Figure 3-d) constructor. This constructor allows two queries to be connected and defines whether the set operation (i.e., Intersect, Union, Except, or Exclusive Union – Figure 3-e) is distinct (["#"]). The right side of Figure 3 illustrates a DSQL query that uses these constructors and shows their respective representations in SQL. This query answers the following question: *"Which employees have no department and which departments have no employees?"*.



```
SELECT employeeId
FROM employee e
WHERE sex = 'M'
AND salary BETWEEN
(SELECT MIN(salary)
   FROM employee
   WHERE sex = 'F')
AND
(SELECT MAX(salary)
   FROM employee
   WHERE sex = 'F');
```

Figure 4. Constructors for *Unary*, *Binary*, and *Ternary* operation.

When all operands of an arithmetic, logical, or comparative operation are subqueries or conditional expressions, the UnaryOperation (Figure 4-a) (e.g., is null), BinaryOperation (Figure 4-b) (e.g., +), or TernaryOperation (Figure 4-c) (e.g., between) constructors must be used. The label for these constructors follows the same syntax as that of the Column constructor. The right side of Figure 4 shows a DSQL query that uses these constructors and their respective representations in SQL. This query answers the following question: *"Who are the male employees with a salary between the minimum and the maximum salary of the female employees?"*.

4. Related studies

We used the ScienceDirect, SpringerLink, ACM, and IEEE Xplore libraries to execute the following search string: "*(SQL AND ("complex query" OR "complex queries" OR subquery OR subqueries) AND (visual OR graphic OR graphical))*". The term *SQL* restricts the search to this language. The terms *complex query*, *complex queries*, *subquery*, and *subqueries* restrict the search to proposals that address complex SQL constructs. The terms *visual*, *graphic*, and *graphical* limit the search to visual approaches. Note that this search string does not consider the terms *join*, *having*, or *case* because they are too broad, nor the term *conditional expression* because they are too restrictive. In the following paragraph, we present a summary of the best three articles found in the literature, because they are the ones that support most complex SQL constructs.

The SIEUFERD [Bakke and Karger 2016] is a form-based VQL that specifies queries by directly manipulating nested relational results, and by using spreadsheet idioms such as formulas and filters, the user can express a relationally complete set of query operators plus calculation, aggregation, outer joins, sorting, and nesting. The GraphSQL [Cerullo and Porta 2007] is a diagram-based VQL that specifies queries using geometric shapes and links: data sources are grids; attributes are circles; conjunctions are rectangles; disjunctions are ellipses; operators are parallelograms; aggregation functions are triangles; grouping and constants are small rectangles; joins are hexagons; and, finally, dotted lines connect data sources with attributes. The ConQuer-II [Bloesch and Halpin 1997] is a hybrid VQL (based on forms and icons) that specifies queries using elementary relationships; operators such as and, or, not, and maybe; contextual for-clauses; and object-correlation, thereby avoiding the need to deal explicitly with implementation details such as relational tables, null values, outer joins, group-by clauses, or correlated subqueries.

A comparative analysis of the expressiveness of DSQL with its principal related studies shows that: (1) all languages support correlated subquery, Inner Join, and Outer Join; (2) only GraphSQL, ConQuer-II and DSQL support non-correlated subquery, grouping restrictions, and subqueries in the Where clause; (3) only SIEUFERD, GraphSQL and DSQL support set operations; (4) only SIEUFERD and DSQL support subqueries in the Select clause; (5) only GraphSQL and DSQL support subqueries in the Having clause; (6) only DSQL supports: conditional expressions (Case/When) and subqueries in the From clause; and (7) no languages support subqueries in the Order By clause. Although no language covers all characteristics, DSQL is the most expressive language because it supports at least 50% more characteristics than its main competitors.

5. Experiment evaluation

The goal of our experiment was to analyze whether DSQL has the potential to be as efficient and easy to understand as SQL. Our experiment focuses on SQL, because the related proposals have drawbacks when compared with DSQL (cf. Section 4) and DSQL is as expressive as SQL (cf. Section 3). In this context, we define the null (H0) and alternative (H1) hypotheses:

H0: *DSQL does not have the potential to be as efficient and easy to understand as SQL.*

H1: *DSQL have the potential to be as efficient and easy to understand as SQL.*

We experimented with 12 undergraduate students from a database course at our university, where seven students had some knowledge of SQL, five students had no knowledge of SQL, and no one knew DSQL. The experimental design was built using a randomized paired comparison design with two treatments on one factor [Wohlin et al. 2012] to avoid bias, where the participants were randomly selected to one of the two teams (A or B). Before starting the experiment, each participant signed a letter of consent[1] and received a 6-hour SQL course[2]. In this course, we presented the DSQL language as an auxiliary tool to help understand SQL concepts. During the experiment, each participant analyzed six complex queries (three SQL + three DSQL[3]) and choose one out of four possible alternatives. That is, only one answer could be selected. We recorded the answers and the time taken for each question. After answering the three questions in each language, participants completed a NASA-TLX questionnaire to collect feedback on their perceptions of the task they had just completed. Furthermore, no answer feedback and no help were given for the participants because this could threaten the validity of the results.

Table 1. Descriptive statistics.

Team	Part	Notation	Questions	Successes	Proportion (Accuracy)	Mean (Time)	S.D. (Time)	S-W (Time)	Mean (Ease)	S.D. (Ease)	S-W (Ease)
A	6	SQL	18	15	0.83	257.94	122.07	.116			
	6	DSQL	18	12	0.67	243.35	86.87	.991			
B	6	SQL	18	8	0.44	249.78	151.43	.124			
	6	DSQL	18	9	0.50	269.73	159.59	.163			
All	12	SQL	36	23	0.64	255.10	129.59	.132	52.3	17.6	.529
	12	DSQL	36	21	0.58	254.65	120.48	.317	49.6	20.9	.571

Table 2. Statistical tests scores

Team	Two-proportion z-test			Welch t-test (Time)			Welch t-test (Ease)		
	Statistic	DF	Significance	Statistic	DF	Significance	Statistic	DF	Significance
A	0.593	1	.441	0.362	24.73	.720			
B	0	1	1	0.264	14.92	.795			
All	0.058	1	.809	0.012	41.983	.991	0.3383	21.391	.738

Table 1 presents the descriptive statistics for the accuracy, time, and ease-of-understanding metrics. The *Part*, *Questions*, and *Successes* columns contain the number of participants, the number of questions these participants answered per notation, and the number of correct answers from these participants, respectively. The *Proportion (accuracy)* column contains the proportion of correct answers (*Successes* divided by *Questions*). The last six columns contain the mean (*Mean*), the standard deviation (*S.D.*), and the *p-value* of the Shapiro-Wilk (*S-W*)) normality tests, the first three related to the mean time of the correct answers and the last three related to the mean ease-of-understanding. Based on the normality tests, we suggest normality on the mean time of the correct answers and on the mean ease-of-understanding ($\alpha = 0.05$).

[1]The letter of consent is available at https://dsqlcase.github.io/experiment/consentletter.pdf

[2]The material is available at https://dsqlcase.github.io/experiment/course.pdf

[3]The questions are available at https://dsqlcase.github.io/experiment/questions.pdf

We used the two-proportion z-test for the accuracy metric and the Welch t-test for the time metric and ease-of-understanding metric [Kitchenham et al. 2017]. Table 2 summarizes the two-proportion z-test. We found no statistical evidence of significant differences between the proportions, including the proportion that corresponds to the sum of successes for all questions (*statistic = 0.058, DF = 1, p = 0.809*). Table 2 also summarizes the Welch t-test results for the time and ease-of-understanding metrics. We found no statistical evidence of significant difference between the mean time and mean ease-of-understanding of the questions (i.e., NASA-TLX workload score).

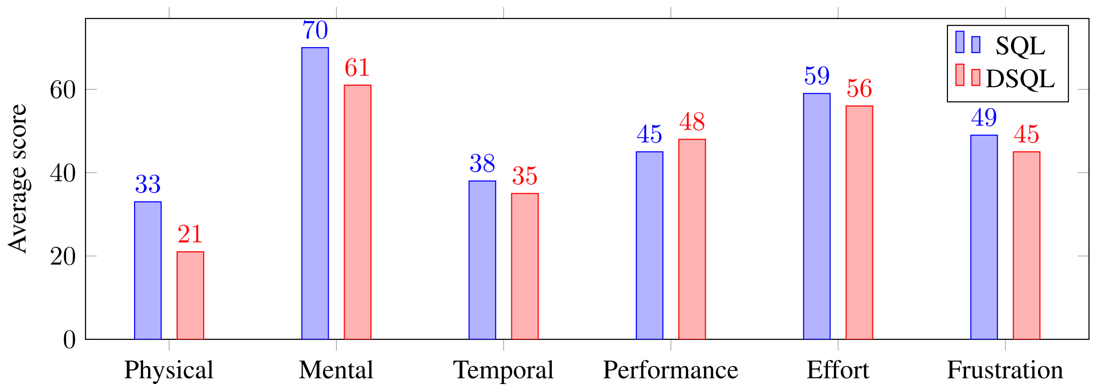


Figure 5. Average score of NASA-TLX dimensions in the scoring phase.

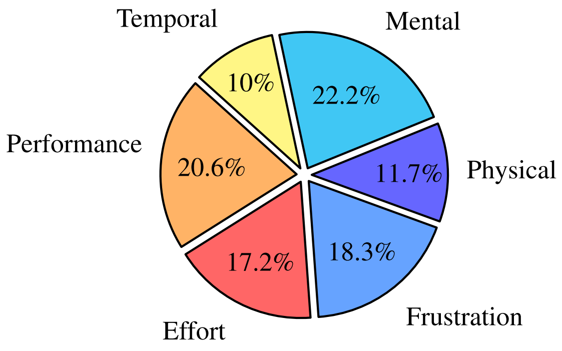


Figure 6. Percentage of the selection of NASA-TLX dimensions in the weighting phase.

The phases of the NASA-TLX protocol (i.e., scoring and weighting – cf. Section 2.2) allowed for interesting observations beyond merely the difference between the mean perceived workload scores. Figure 5 shows the average score of the NASA-TLX dimensions in the scoring phase. Note that mental demand is the dimension with the highest average score, followed by effort, frustration, performance, temporal demand, and, finally, physical demand. DSQL have lower workload average scores in most dimensions (i.e., physical demand, mental demand, temporal demand, effort, and frustration); and SQL has a lower workload average score only in the performance dimension. Figure 6 shows the percentage of the selection of NASA-TLX dimensions in the weighting phase. Note that mental demand is the dimension with the highest percentage (22.2%), followed by performance (20.6%), frustration (18.3%), effort (17.2%), physical demand (11.7%), and, finally, temporal demand (10%). The analysis of both phases suggests that mental demand is perceived as the most demanding dimension and the most relevant dimension in the workload, while temporal demand and physical demand are perceived as the least demanding and least relevant dimensions in the workload.

Our results suggest that there was no observable benefit in the efficiency and easy of understand, because the participants were able to conduct their tasks using SQL and DSQL. One possible reason for this equivalence is because SQL (declarative) and DSQL (UML based) notations are easy to understand. According to [Eitrheim and Fernandes 2016], workload levels below 50 were perceived as acceptable, and both SQL and DSQL had workloads close to that level (i.e., 52.3 and 49.6 respectively). In this context, it is important to highlight that 58% of the students had some knowledge of SQL and 0% of the students had knowledge of DSQL. Even with this unfavorable context for DSQL, we can reject the null hypothesis and accept the alternative hypothesis. That is, that DSQL has the potential to be as efficient and easy to understand as SQL.

6. Conclusion

Given the academic and industrial relevance of SQL and the potentially complex nature of queries written in this language, VQLs that favor its use and, consequently, decrease the MW are important pedagogical and technological tools. In this context, we investigated the MW impact of a visual query language (i.e., DSQL) on the understanding of SQL queries. In an experiment, we compared the accuracy, the time, and the ease-to-understanding complex queries using both languages. The results indicate that there is no significant difference, but that DSQL was slightly faster and easier to understand than SQL. The NASA-TLX protocol was used to measure the MW impact. According to [Eitrheim and Fernandes 2016], the perceived workload score for both languages are acceptable (i.e., ≤ 50). In addition, DSQL has lower workload average scores than SQL in almost all analyzed NASA-TLX dimensions (i.e., physical demand, mental demand, temporal demand, effort, and frustration), with SQL being better only in the performance dimension. We highlight that the visual notation of DSQL favors the specification of complex SQL queries because it uses: (1) modularized representation that favors the focus and reuse of specific parts of the query; (2) visual abstractions that are easily extendable from links; and (3) better coverage of complex SQL constructs (at least 50% more expressive than its main competitors). In summary, our experimental evaluation gave evidence that DSQL has the potential to be as understandable and efficient as SQL without increasing the MW and that DSQL advances the state of the art of VQLs because it mitigates the drawbacks of related proposals. Therefore, DSQL can be an alternative for users that prefer to work with a visual notation instead of textual syntax.

We only addressed the MW impact on understanding SQL queries in this study. Further studies should consider the MW impact on specifying SQL queries, as well the interactions between understanding and specifying.

References

Abouzied, A., Hellerstein, J., and Silberschatz, A. (2012). Dataplay: Interactive tweaking and example-driven correction of graphical database queries. In *Proc. of ACM Symp. on User Interface Software and Technology*, p. 207–218, New York, NY, USA. ACM.

ACM/IEEE (2013). *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM, NY, USA. 999133.

ACM/IEEE (2017). *Information Technology Curricula 2017: Curriculum Guidelines for Baccalaureate Degree Programs in Information Technology*. ACM, NY, USA.

Bakke, E. and Karger, D. R. (2016). Expressive query construction through direct manipulation of nested relational results. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, p. 1377–1392, New York, NY, USA. ACM.

Bloesch, A. C. and Halpin, T. A. (1997). Conceptual queries using conquer-ii. In *International Conference on Conceptual Modeling*, p. 113–126. Springer.

Brambilla, M., Cabot, J., and Wimmer, M. (2017). Model-Driven Software Engineering in Practice: Second Edition. *Synthesis Lectures on Software Engineering*, 3(1):1–207.

Cain, B. (2007). A review of the mental workload literature. Technical report, Defence Research And Development Toronto (Canada).

Catarci, T. and Santucci, G. (1995). *Diagrammatic Vs Textual Query Languages: A Comparative Experiment*, p. 69–83. Springer US, Boston, MA.

Cerullo, C. and Porta, M. (2007). A system for database visual querying and query visualization: Complementing text and graphics to increase expressiveness. In *Inter. Workshop on Database and Expert Systems Applications*, p. 109–113.

Czejdo, B. D., Tucci, R. P., Embley, D. W., and Liddle, S. W. (1993). Graphical query specification with participation constraints. In *Computing and Information, 1993. Proceedings ICCI '93., Fifth International Conference on*, p. 433–437.

Eitrheim, M. H. and Fernandes, A. (2016). The nasa task load index for rating workload acceptability. In *Human Factors and User Needs in Transport, Control, and the Workplace*.

El-Mahgary, S. and Soisalon-Soininen, E. (2015). A form-based query interface for complex queries. *J. Vis. Lang. Comput.*, 29(C):15–53.

Gryz, J., Wang, Q., Qian, X., and Zuzarte, C. (2008). Sql queries with case expressions. In *Foundations of Intelligent Systems*, p. 351–360, Berlin, Heidelberg.

Hart, S. G. and Staveland, L. E. (1988). Development of nasa-tlx (task load index): Results of empirical and theoretical research. In Hancock, P. A. and Meshkati, N., editors, *Human Mental Workload*, volume 52 of *Advances in Psychology*, p. 139 – 183. North-Holland.

Hvorecký, J., Drlík, M., and Munk, M. (2010). The effect of visual query languages on the improvement of information retrieval skills. *Procedia - Social and Behavioral Sciences*, 2(2):717 – 723. Innovation and Creativity in Education.

Kawash, J. (2004). Complex quantification in structured query language (sql): A tutorial using relational calculus. *Journal of Computers in Mathematics and Science Teaching*, 23(2):169–190.

Kitchenham, B., Madeyski, L., Budgen, D., Keung, J., Brereton, P., Charters, S., Gibbs, S., and Pohthong, A. (2017). Robust statistical methods for empirical software engineering. *Empirical Software Engineering*, 22(2):579–630.

Longo, L. (2015). A defeasible reasoning framework for human mental workload representation and assessment. *Behaviour & Information Technology*, 34(8):758–786.

StackOverflow (2018). Stack overflow developer survey results 2018. https://insights.stackoverflow.com/survey/2018. Accessed: 2018-12-14.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012). *Experimentation in software engineering*. Springer Science & Business Media.

Zhang, G., Chu, W. W., Meng, F., and Kong, G. (1999). Query formulation from high-level concepts for relational databases. In *Proc. User Interfaces to Data Intensive Systems*.