Challenges with Teaching and Learning Theoretical Query Languages

Jalal Kawash and Levi Meston
Department of Computer Science, University of Calgary, 2500 University Dr. NW, Calgary, Alberta, Canada

Keywords: Computer Science Education, Tools for Education, Database Systems, SQL.

Abstract: Relational algebra and relational calculus are often taught as part of the topics of a typical database course in Computer Science and Engineering degrees. The teaching of these theoretical languages not only provide the students with the theoretical foundation for learning SQL, but also serve as a vehicle for students to sharpen their problem solving skills. SQL, the most commonly used database language, also serves as a substantial part of any course on databases. SQL was supposed to be an implementation of relational calculus; however, the language ended up being a hybrid implementation of both the calculus and the algebra. One challenge that faces students and educators alike is that unlike SQL where queries can be tested and validated with existing databases, the calculus and the algebra remain theoretical with very limited support for such testing or validation. In addition, not all theoretical constructs in both the algebra and the calculus have a straightforward implementation in SQL. After discussing these challenges, we make the case in this paper for the need to better computer tools that support the teaching and learning of these two theoretical query languages. We also present the features/objectives of such a tool that we are currently developing.

1 INTRODUCTION

Computer Science, Software Engineering, and similar programs include database systems as a core topic that students need to master before they are ready for their future jobs. The Structured Query Language (SQL) constitutes a substantial part of any course on database systems. It is by far the most commonly used query language in Relational Database Management System implementations. Oracle, MySQL, SQL Server, and Microsoft Access are just a few examples. Hence, it is not a surprise that database textbooks allocate a substantial space for the topic. Moreover, many introductory textbooks on database management (such as (Elmasri and Navathe, 2015; Silberschatz et al., 2019; Connolly and Begg, 2014)) also includes discussion of what is called *theoretical query languages*.

These theoretical query languages are *relational calculus* and *relational algebra*. The calculus has two types: the domain calculus and the tuple calculus. The former is not very popular; so, we restrict the discussion in this paper to the tuple relational calculus. Furthermore, the concepts in these two calculi are the same, but the range of calculus variables is what makes them different: one ranges over domains (columns) and the other over tuples (rows). Historically, SQL was meant to be an informal rendering of tuple relational calculus (Codd, 1972). Yet, anyone can easily see that SQL turned out to be a hybrid implementation of both the calculus and the algebra.

The calculus and the algebra have opposite design philosophies. The former is non-procedural or declarative, where queries are expressed using predicate logic to define the set of tuples that satisfy the query condition. Hence, the way the query is formulated does not influence the way it is executed. The algebra is procedural and the way an algebra routine is written affects how it is executed.

McMaster et al. make the case for teaching these theoretical languages in database courses by discussing several benefits to students (Mcmaster et al., 2008). There are three major benefits for learning these two theoretical languages. First, they provide a theoretical mechanism for students to develop their problem solving skills, especially in the query formulation domain. Second, they provide a theoretical foundation for learning SQL. Finally, they provide an opportunity to contrast procedural versus non-procedural query language design, allowing for important discussion on query processing. However as noted by McMaster et al. (Mcmaster et al., 2008), the coverage of these languages in leading database textbooks (such as (Elmasri and Navathe, 2015; Sil-

Kawash, J. and Meston, L.
Challenges with Teaching and Learning Theoretical Query Languages.
DOI: 10.5220/0009471103820389
In *Proceedings of the 12th International Conference on Computer Supported Education (CSEDU 2020) - Volume 2*, pages 382-389
ISBN: 978-989-758-417-6

berschatz et al., 2019; Connolly and Begg, 2014)) is purely mathematical and the provided problems are "pencil and paper" exercises rather than being of the programming nature.

Our experience is consistent with others (Kessler et al., 2019; Mcmaster et al., 2008) and shows that students struggle with these languages due to the lack of computer tools that can validate their queries. In SQL, the students can test their SQL scripts against a concrete database and verify the correctness of their code. However, this luxury is not often available with the calculus or the algebra. Our future objective is to provide a Web-enabled tool that utilizes a visual program language(Guo et al., 2008) in order to allow users to (visually) formulate calculus and algebra query statements. The system translates these statements to SQL and then validates them through execution against a concrete database. This intermediate step consisting of translating algebra and calculus statements to SQL is crucial for achieving the most benefits of learning these languages through drawing the relationship with practical database programming, typically with SQL as a major component.

In this paper, we present grade and survey analysis to support the claim that these topics are challenging to students. We also analyze qualitative feedback from students that point at the lack of computer-based tools which can help the student learn these languages better by validating and executing their theoretical queries against concrete databases, much like SQL. Another conclusion form this feedback is often the lack of ability to associate algebra and calculus queries with SQL adds to their learning challenges. Our future tool is designed to answer all of these challenges.

Some previous tools have been implemented to address this shortcoming of theoretical query languages (Kessler et al., 2019; Eckberg, ; Yang, ; Muehe, ; Kawash, 2014; Mcmaster et al., 2008). There are shortcomings of these tools as we will discuss in Section 6. First, we provide some necessary technical background about these languages in Section 2. A brief description of the course in which the data is collected and analyzed in given in Section 3. Challenges faced by students are discussed in Section 5.The requirements of our future tool are described in Section 7 (we do not describe the tool in this paper) and Section 8 concludes the paper.

2 THEORETICAL QUERY LANGUAGES

2.1 Relational Calculus

Relational calculus (RC) is a non-procedural, declarative language. A query statement is a specification of the resulting set from a query using predicate logic. Hence, it relies on the existential ($\exists$) and universal quantifiers ($\forall$) to describe this set. SQL has a direct support for the existential quantifier ($\exists$) through its `EXISTS` function. However, universal quantifiers ($\forall$) are not directly supported in SQL and must be also expressed in terms of `EXISTS`. This is due to the fact that the predicate $\forall xP(x)$ is equivalent to the predicate $\neg\exists x\neg P(x)$. This is normally a huge source for confusion for students learning SQL (Kawash, 2014). A hypothetical RC statement of the following form:

$\{x|A(x) \wedge \forall y(B(y) \rightarrow x.a = y.b)\}$

translates to the following equivalent statement:

$\{x|A(x) \wedge \neg\exists y(B(y) \wedge x.a \neq y.b)\}$,

and the latter translates to the following equivalent SQL query:

```
SELECT * FROM A WHERE NOT EXISTS
(SELECT * FROM B WHERE A.a != B.b)
```

We provide an example query in relational calculus to highlight the difficulties with formulating such queries and constructing their equivalent SQL code. Our example assumes the following relations:

Species*(sname, maxht)*
Tree*(tid, yplanted)*
Measurement*(mid, mdate, tid, height, nbranches)*

A calculus statement to *retrieve the name for each species with a maximum height exceeding 20 meters such that every tree of that species that was planted before 1950 has had every measurement showing a height greater than 10 meters* would be:

$\{s.sname|Species(s) \wedge s.maxht > 20 \wedge (\forall t)((Tree(t) \wedge t.sname = s.sname \wedge t.yplanted < 1950) \rightarrow (\forall m)((Measurement(m) \wedge m.tid = t.tid) \rightarrow m.height > 10))\}$

This statement needs to be transformed to a form that does not contain universal quantifiers or implication to simplify the SQL translation. As mentioned earlier, the predicates of the form $\forall xP(x)$ are replaced by the form $\neg\exists x\neg P(x)$. An implication $a \rightarrow b$ is replaced by $\neg a \vee b$. The end result is:

$\{s.sname | Species(s) \wedge s.maxht > 20 \wedge (\neg\exists t)((Tree(t) \wedge t.sname = s.sname \wedge t.yplanted < 1950) \wedge (\exists m)((Measurement(m) \wedge m.tid = t.tid) \wedge m.height \leq 10))\}$

Now, this can be translated to SQL as follows:

```
SELECT s.sname
FROM Species AS s
WHERE s.maxht > 20
AND NOT EXISTS(
            SELECT *
            FROM Tree AS t
            WHERE t.sname = s.sname
            AND t.yplanted < 1950
            AND EXISTS(
                        SELECT *
                        FROM Measurement AS m
                        WHERE m.tid = t.tid
                        AND m.height <= 10) )
```

2.2 Relational Algebra

All of the relational algebra operations except division ($\div$) are directly supported in SQL. These are summarized in Table 1.

Let R_1 and R_2 be relations. Meant to capture a restricted form of universal quantification, the division operation $R_1(Z) \div R_2(X)$ requires the sets of attributes Z and X to have the property $X \subseteq Z$. Let $Y = Z - X = \{a_1, a_2, \cdots, a_n\}$, $R_1(Z) \div R_2(X)$ can be written in SQL as follows:

```
SELECT Y FROM R1
WHERE NOT EXISTS
               (SELECT * FROM R2
                WHERE R1.a1 != R2.a1
                OR R1.a2 != R2.a2
                ...
                OR R1.an != R2.an )
```

Alternatively, $R_1(Z) \div R_2(X)$ can be expressed using the following algebra routine, and this routine can be translated to SQL. In either case, division queries are a source of struggle for most students.

$Res_1 \leftarrow \pi_Y(R_1)$
$Res_2 \leftarrow \pi_Y((R_2 \times Res_1) - R_1)$
$Result \leftarrow Res_1 - Res_2$

Finally, the assignment statement ($\leftarrow$) can be implemented in SQL in different ways. One obvious way is to use virtual tables, or *views*. For instance, the previous algebra routine can be written in SQL as:

```
CREATE VIEW Res1 AS
SELECT Y FROM R1;

CREATE VIEW tmp AS
(
SELECT * FROM R2, Res1
MINUS
SELECT * FROM R1
);

CREATE VIEW Res2 AS
SELECT Y FROM tmp;

SELECT * FROM Res1
MINUS
SELECT * FROM Res2;
```

3 THE COURSE

The context of this study is a third-year elective course in Database Systems at the University of Calgary. The main audience for this course are from Computer Science and Software Engineering. Other students, such as Bioinformatics majors, also take this course. The course constitutes of six modules:

1. Introduction: 1 unit
2. Entity Relationship (ER) and Enhanced Entity Relationship (EER) models: 2 units
3. Relational model and Mapping ER and EER models to the relational model: 3 units
4. Relational Algebra (RA) and Relational Calculus (RC): 3 units
5. SQL: 4 units
6. Normal forms and decomposition (NF): 2 units

Each unit is equivalent to 75 minutes of lectures and 50 minutes of tutorials. The latter are conducted by teaching assistants. The six modules sum up to 15 units. However, a semester is 11 weeks, allowing for 22 units. The remaining 7 units are reserved for a form of in-class group activities to re-enforce the material in each of modules 2 to 6. We call these group exams graded group activities (GGAs). Each module is concluded with a take-home assignment to be performed individually by students. There is a final exam and a group project due at the end of the course. The group project is irrelevant to this study and will not be discussed further. The focus of this paper are modules 3 and 4 dealing with theoretical

Table 1: Relational Algebra Operations.

Name	Algebra Operation	SQL Implementation
Select	$\sigma_{<condition>}(R)$	`SELECT * FROM` R `WHERE` <*condition*>
Project	$\pi_{<attributes>}(R)$	`SELECT` <*attributes*> `FROM` R
Theta Join	$R_1 \bowtie_{<condition>} R_2$	`SELECT` $*$ `FROM` R_1 `INNER JOIN ON` R_2 `WHERE` <*condition*>
Natural Join	$R_1 *_{<condition>} R_2$	`SELECT` $*$ `FROM` R_1 `NATURAL JOIN ON` R_2 `WHERE` <*condition*>
Left Outer Join	$R_1 ⟕_{<condition>} R_2$	`SELECT` $*$ `FROM` R_1 `LEFT OUTER JOIN ON` R_2 `WHERE` <*condition*>
Right Outer Join	$R_1 ⟖_{<condition>} R_2$	`SELECT` $*$ `FROM` R_1 `RIGHT OUTER JOIN ON` R_2 `WHERE` <*condition*>
Full Outer Join	$R_1 ⟗_{<condition>} R_2$	`SELECT` $*$ `FROM` R_1 `FULL OUTER JOIN ON` R_2 `WHERE` <*condition*>
Union	$R_1 \cup R_2$	`SELECT` $*$ `FROM` R_1 `UNION SELECT` $*$ `FROM` R_2
Intersection	$R_1 \cap R_2$	`SELECT` $*$ `FROM` R_1 `INTERSECT SELECT` $*$ `FROM` R_2
Difference	$R_1 - R_2$	`SELECT` $*$ `FROM` R_1 `MINUS SELECT` $*$ `FROM` R_2
Cartesian Product	$R_1 \times R_2$	`SELECT` $*$ `FROM` R_1, R_2
Division	$R_1 \div R_2$	No direct implementation
Rename	$\rho_{S(a_1,a_2,\cdots,a_n)}$, ρ_S, or $\rho_{(a_1,a_2,\cdots,a_n)}$	Implemented through `AS`

query languages. Specifically in the next section, we will support our thesis that student struggle in these modules by comparing the GGA, assignment, and final exam marks for various modules or topics. GGAs have two stages. In the first stage, each student individually attempts a set problems. Then, they team up to attempt the same set of problems as a group. We already discussed GGAs in a previous paper (Kawash et al., 2020).

4 GRADE ANALYSIS

We analyzed the grades from 125 students enrolled in one semester in this course. It is worth noting that students are given a full week to individually complete each assignment, and these assignments constitute a collection of constructive-response questions. For RA/RC and SQL assignments, each consists of 10 English query statements that need to be written in RA, RC, and SQL. During the duration of an assignment, the students have access to feedback from the instructor and TA on their attempts. GGAs are of the multiple choice format, and they utilize immediate feedback instruments in the form of "scratchies" (ifa,). Students scratch a card to discover if their answers were correct, allowing multiple-attempts for each question. This yes-no feedback is characteristically different from the feedback that can be received for attempts to answer assignment questions. With assignments, students can have lengthy discussions with the instructor or the TA over why or why not a certain attempt is an acceptable solution. Such lengthy feedback is available for GGAs only after the fact and not while taking the group exam. For the final exam, the questions are of the constructive-response type, and the students do not have any opportunity for feedback during the exam.

Average grades from the assignments are depicted in Figure 1. The standard deviations are 7.77 (ER), 18.89 (Relational), 16.72 (RA/RC), 11.42 (SQL), and 6.71 (NF). While the RA/RC assignment has the lowest average, there is no substantial difference between the RA/RC assignment and the rest of the topics. We emphasize the difference between the averages of the RA/RC and SQL assignment. These two assignments are of the same nature: the students are asked to formulate queries. However, SQL queries can be validated using any database management system, while in the theoretical languages assignment, the only feedback available to students is that given when they seek it from the instructor or the TA.

The story is slightly different for GGAs. The RA/RC GGA records the lowest score compared to the remaining GGAs as is shown in Figure 2. This is in spite of the exploitation of the limited immedi-

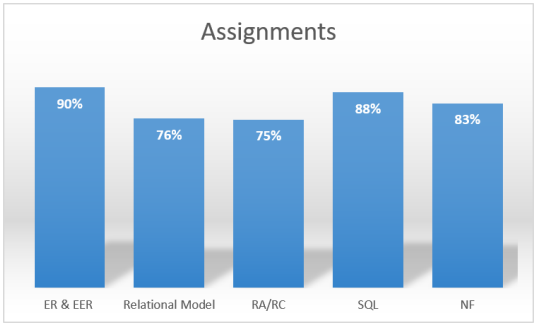

Figure 1: Comparison of average marks for different assignments.

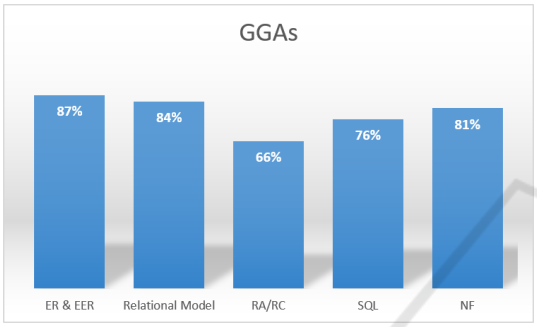

Figure 2: Comparison of average marks for different GGAs.

ate feedback mechanism. The standard deviations are 8.47 (ER), 10.28 (Relational), 13.79 (RA/RC), 18.35 (SQL), and 24.11 (NF).

When feedback disappears completely such as in the final exam, Figure 3 shows a substantial dip in the RA and RC exam questions. We separated these questions into three categories: RA for relational algebra questions, RC for relational calculus questions, and RA/RC is for questions that combine both. The standard deviations are 18.76 (ER/Relational), 29.17 (RA), 28.86 (RC), 30.42 (RA/RC), 16.97 (SQL), and 21.55 (NF). In all of these categories, the average is at least 32% points lower than other categories. Contrast these averages with that of SQL questions. Theoretical query questions and SQL questions are of exactly the same type (constructive-response) and they are at the same level of complexity. Yet, we see that the students are doing better in SQL questions, in spite of the absence of immediate feedback during the exam.

We conjecture that students are better at formulating queries in SQL than formulating these in RA or RC is due to their ability to practice and validate these queries through executing them against a DBMS implementation. Such an opportunity is not available for RA and RC queries.

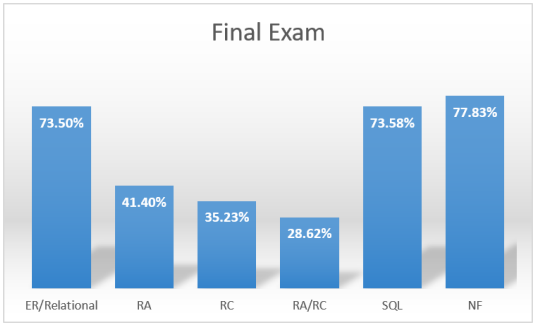

Figure 3: Comparison of average marks for final exam questions grouped by topic.

5 STUDENT FEEDBACK

In a previous study, we asked the students to create an educational video about topics that they thought were challenging or ones they struggled with (Kawash and Sailunaz, 2020). A summary of the chosen topics by all 97 students who took part in this experiment is shown in Figure 4. A total of 46% of surveyed students identified RA and RC as challenging topics compared with 28% for SQL. In addition, 13% identified subjects that are common to RA, RC, and SQL as challenging. Hence, a total of 59% of the surveyed students selected subjects from theoretical languages.

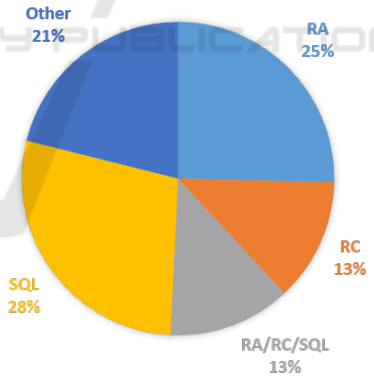

Figure 4: Difficult topics as identified by students.

Our anonymous post-semester feedback from students gives us insights as to why the students find RA and RC to be difficult subjects. This feedback has two converging themes:

1. **Lack of Opportunity for Validation:** Students constantly reported their frustration with the lack of tools that allow them to validate and test the correctness of their queries. The only way to validate their queries is to refer them to the instructor or the teaching assistant for feedback. How-

ever, there is often a large turn-around time for this feedback. They expressed their desire for a tool that provides this feedback instantaneously as is the case when testing SQL queries.

2. **Challenge for Associating Complex Theoretical Queries with SQL:** Students stressed in their feedback the importance of the ability to associate theoretical query statements with SQL statements. This was particularly true for more complex queries. After all, simple queries may not justify the development of tools to validate them. These are needed more when the logic becomes complex. Students argued that the ability to associate or map theoretical queries to SQL not only allowed them to feel more confident about their understanding of the subject matter, but also thought that it forced them into reflecting on all three language design philosophies. Some students reported that this also opened their eyes to the subject of query processing.

We have shown that students struggle in the RA and RC topics more than any other topic in the course, including SQL. The marks in RA/RC assessment is constantly lower regardless of the incorporated assessment method. In addition, student feedback confirmed the challenges they face when learning these theocratic languages. Hence, we believe that we have made the case for the need of a computer-based tool that allow the students to validate and execute their RA and RC queries.

6 EXISTING TOOLS

Previous tools have been implemented to address this shortcoming of theoretical query language validation. McMaster et al. (Mcmaster et al., 2008) describe two programming approaches for the algebra and the calculus. For the first, they developed a function library using Visual FoxPro. For the latter, a Prolog library is implemented. While they argue it is minimal, some knowledge in FoxPro and Prolog is required from students to use these libraries. In addition, a setup overhead, such as defining the predicates that represent a database mode in Prolog, is required from students. In both cases, for RA and RC, no equivalent SQL code is generated.

IRA (Muehe,) is a web-enabled tool and RA (Yang,) is a stand-alone application. Both are limited to RA and do not support RC. They also have other limitations, including lack of support to some important algebra operators, limited data sets, and limited user interface capabilities. Both IRA and RA generate an equivalent SQL statement.

RelaX (Kessler et al., 2019) is also dedicated to RA, and it is Web-based. However, it simply provides the result set of the query statement. It does not provide an equivalent SQL statement for the algebra statement as an intermediate step. Seeing the relationship between these two statements is beneficial for the understanding of both statements as the student feedback indicated. Some of the relational algebra constructs are directly supported in SQL, but others have no direct support. Understanding how to formulate such a query in SQL can be cumbersome for students. The division operation is one such example.

There are even fewer tools for the calculus. Calculus Emulator (Eckberg,) is a stand-alone application that generates the equivalent SQL code. QuantX (Kawash, 2014) is also a stand-alone application, but it simply teaches the users how to translate a complex calculus query to SQL without providing any validation.

Finally none of these tools uses a visual programming language approach, such as tile-based programming (Guo et al., 2008). In addition, none of them provide support for quizzes and challenges and they do not collect information about their pattern of usage either. Such information can be valuable for instructors and researchers alike.

7 OUR PROPOSED TOOL

We are currently developing a tool that allows students to develop and validate theoretical queries, answering all the limitations of previous tools. In summary, the tool:

1. **Is Web-based:** Unlike most existing tools (RA, calculusEM, & QuantX), our tool is a Web-based application, alleviating any complexities from the user the need to install it or to install any required packages.
2. **Validates Queries against Any Database:** Most previous tools limit the data sets a user can work with (IRA, RA, QuantX). Our tool allows the user to incorporate any existing database or to create a new database from scratch. Hence, students can work with any examples presented in their lectures, course material, or textbook.
3. **Supports Both Algebra and Calculus:** Most existing tools support one of the two theoretical query languages, but not both. Our tool has support for both relational algebra and relational calculus. In addition, it has support for all operations of these two languages.

4. **Generates SQL:** Based on student feedback, it is beneficial to see the correspondence between an algebra or calculus query with the SQL query. Our tool provides the corresponding query, in addition to the result set once the query is executed against a database instance. This is especially important for complex queries, where the relationship between the theoretical query and the SQL statement is not straight forward. In Section 2, we have provided some examples of such complex queries.
5. **Uses Tile-based Programming:** The tool utilizes visual programming, through tile-based programming. No other tool provides this feature. Tile-based programming does not only make it easier for students to write their programs, but also has many other beneficial side effects (Guo et al., 2008). Similar to other algorithms, algebra and calculus can be expressed naturally using tiles. Tiles also allow for better execution performance because the approach a very effective way to exploit locality (Guo et al., 2008). Finally, they can also be used to express data distribution and parallel computation (Guo et al., 2008). In one word, they provide students with exposure to concepts outside the database topics.
6. **Supports Customized Quizzes and Challenges:** The tool allows educators and researchers to create quizzes and challenges for students.
7. **Collects and Logs Usage Information:** Unlike all other tools, the tool collects and logs all types of information for educators and researchers to analyze. It will help educators and researchers look for certain patterns, identifying student weaknesses, areas of struggle, and henceforth.

8 CONCLUSION

Learning theoretical query languages has benefits to students since they help improve problem solving skills. In addition, both relational algebra and relational calculus provide the theoretical foundation for the design of SQL. Mastering these languages improves the skills needed to formulate SQL queries, specifically for queries that incorporate challenging logic. Unlike SQL, a common challenge that impedes the teaching and learning of these theoretical query languages is the limited opportunity for students to execute and validate their query statements against an existing database. In addition, students indicated that it would be more beneficial to them if they can also see a corresponding SQL code to the queries formulated in the theoretical languages. To the best of our knowledge, there are very few existing tools that support the execution of either algebra or calculus queries. These tools have many limitations, and we are developing a tool that answers all of these limitations. Namely, our tool is Web-based, supports both languages, not specific to a fixed database, generates SQL for comparison purposes, utilizes tile-based programming, and creates analysis logs.

ACKNOWLEDGMENTS

We are thankful to the anonymous referees. Their feedback helped us improve the paper.

REFERENCES

Epstein Educational Enterprises: What is IF-AT? http://www.epsteineducation.com/home/about/. Accessed: 2019-07-30.

Codd, E. F. (1972). Relational completeness of data base sublanguages. *IBM Research Report*, RJ987.

Connolly, T. and Begg, C. E. (2014). *Database Systems: A Practical Approach to Design, Implementation and Management*. Pearson, USA, 6th edition.

Eckberg, C. Relational Calculus Emulator. https://edoras.sdsu.edu/~eckberg/relationalcalculusemulator.html. Accessed: 2020-01-04.

Elmasri, R. and Navathe, S. B. (2015). *Fundamentals of Database Systems*. Pearson, 7th edition.

Guo, J., Bikshandi, G., Fraguela, B. B., Garzaran, M. J., and Padua, D. (2008). Programming with tiles. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP'08, page 111–122, New York, NY, USA. Association for Computing Machinery.

Kawash, J. (2014). Formulating second-order logic conditions in SQL. In *Proceedings of the 15th Annual Conference on Information Technology Education, SIGITE 2014, Atlanta, Georgia, USA, October 15-18, 2014*, pages 115–120.

Kawash, J., Jarada, T., and Moshirpour, M. (2020). Group exams as learning tools: Evidence from an undergraduate database course. In *SIGCSE '20: The 51st ACM Technical Symposium on Computer Science Education, Portland, OR, USA, March 11-14, 2020*, pages 626–632.

Kawash, J. and Sailunaz, K. (2020). Learning by creating instructional videos: An experience report from a database course. In *Proceedings of EDUCON 2020 – IEEE Global Engineering Education Conference*. IEEE CS Press.

Kessler, J., Tschuggnall, M., and Specht, G. (2019). Relax: A webbased execution and learning tool for relational algebra. In Grust, T., Naumann, F., Böhm, A.,

Lehner, W., Härder, T., Rahm, E., Heuer, A., Klettke, M., and Meyer, H., editors, *BTW 2019*, pages 503–506. Gesellschaft für Informatik, Bonn.

Mcmaster, K., Anderson, N., and Blake, A. (2008). Teaching relational algebra and relational calculus: A programming approach. *Information Systems Education Journal*.

Muehe, H. IRA - interaktive relationale algebra. http://db.in.tum.de/people/sites/muehe/ira/. Accessed: 2020-01-04.

Silberschatz, A., Korth, H. F., and Sudershan, S. (2019). *Database System Concepts*. McGraw-Hill, Inc., USA, 7th edition.

Yang, J. RA (radb). https://users.cs.duke.edu/~junyang/radb/. Accessed: 2020-01-04.