BR∀CE: A Tool for Supporting the Teaching and Learning of Database Theoretical Query Languages through Composing Tiles

Jalal Kawash and Levi Meston

Abstract— **Learning theoretical query languages, namely rela-tional algebra and calculus, is beneficial to students. However, it is often challenging for both students and educators since textbook coverage of the area is purely theoretical with "pencil--and-paper" exercises. Hence, there is no mechanism to validate student queries using a database management system or by automatic translation of these theoretical expressions to an actual database query language like SQL. To answer these limitations, we developed a Web-based tool called BR∀CE that allows users to visually formulate relational algebra and calculus queries in a Scratch-like, tile-based manner. BR∀CE translates the visual query to the equivalent theoretical expression and it also generates an equivalent SQL expression. To use the tool, a user simply needs a browser. BR∀CE also allows the user to work with any database schema. These factors coupled with our plan to make it freely available online make it an important resource for students and educators alike. This paper walks readers through BR∀CE to formulate relational algebra and relational calculus queries.**

Index Terms— **Database Education; Queries; Relational Alge-bra; Relational Calculus; SQL**

I. Introduction

Computer Science, Software Engineering, Information Systems, 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 (RDMS) implementations. Oracle, MySQL, SQL Server, and Microsoft Access are just a few examples of RDBMS implementations that utilize SQL. Hence, it is not a surprise that database textbooks allocate a substantial space for the topic. In addition to SQL, many textbooks also dedicate space for theoretical query languages (such as [3], [8], [17]). There are two such languages: Relational Algebra (RA) and Relational Calculus (RC) [2]. SQL was intended to be an implementation of RC [2]; however, the language ended up being a hybrid implementation of both the RC and RA.

RA is a procedural language, where the way the query statement is defined dictates how the query is executed. Most RA operations were implemented into SQL. These include selection, projection, joins, set operations, and aggregate func-tions. RA also includes a division operation, which was never implemented in SQL. RC, on the other hand, is non-procedural or declarative, where a query statement describes the result set of the query. Hence, the query statement does not dictate how the query is executed. RC has two types: The Tuple-RC and the Domain-RC. Both types of RC are similar, and they use predicate logic to define the result set of the query statement.

Therefore, they both use quantifiers: existential (∃) and universal (∀). The only difference between these two relational calculi is the range of the predicate variables, which range over tuples in the former and domains in the latter. RC queries in BR∀CE are restricted to Tuple-RC since it is the more popular calculus.

SQL has direct support for the existential quantifier (∃) through its EXISTS function. However, universal quantifiers (∀) are not directly supported and must be also expressed in terms of EXISTS, exploiting the fact that the proposition $\forall x P(x)$ is equivalent to $\neg\exists x \neg P(x)$. This is normally a source for confusion and struggle for students [11].

Teaching and learning these theoretical query languages has many benefits to students [14]. We list three major and obvious benefits. First, they provide a theoretical platform for students to develop and sharpen their problem-solving skills, especially in the query formulation domain. These skills are necessary regardless of the language employed. Second, they provide a theoretical vehicle for learning SQL. SQL combines design elements from both RC and RA. Finally, they provide an opportunity to contrast procedural, as it is the case with RA, versus non-procedural query language design, such as in RC, allowing for important refection on language design philosophies as well as query processing.

The coverage of RA and RC in textbooks is purely mathematical and the provided problems are "pencil-and-paper" exercises, as noted by McMaster et al. [14], rather than being of the programming nature. This adds an extra layer of com-plexity to an already complex and unpopular subject among students. Our experience is consistent with others [12]–[14] and shows that students struggle with these languages in part due to the lack of computer tools that can support them with their learning. In SQL, the students can test and validate their SQL code using a database management system, and therefore, verify the correctness of their code. However, this feature is not easily available with RC or RA. This impacts how and if this subject is taught at all. Instructors can easily shy away from it unless they can provide appropriate support for students to check and validate their on-paper answers. Another challenge that faces students is that it is often cumbersome to relate RA or RC queries with SQL [12], especially for involved and complex queries.

To answer these limitations, we developed a Web-based tool called BR∀CE (short for Blockly Relational Algebra and Calculus Environment) that allows students to construct visual, tile-based queries in both RA and RC (See Figure 1). That is, the queries are written in a Scratch-like fashion. The tool generates the equivalent RA or RC expression to the visual query and translates the RA or RC expression to SQL as well. This provides the opportunity for students to validate their queries, through validating the SQL statement as well as the opportunity to better relate these theoretical languages with SQL. BR∀CE is available along with the support resources at the URL https://www.cpsc.ucalgary.ca/~jkawash/brace.html.

Received March 14, 2021, Accepted May 20, 2021, Published online June 27, 2020

Jalal Kawash is with the Department of Computer Science at the University of Calgary, 2500 University Dr. NW, Calgary Alberta, Canada T2N1N4 (e-mail: jalal.kawash@ucalgary.ca).

Levi Meston is with the Department of Computer Science at the University of Calgary, 2500 University Dr. NW, Calgary Alberta, Canada T2N1N4 (e-mail: Levi.Meston@ucalgary.ca).

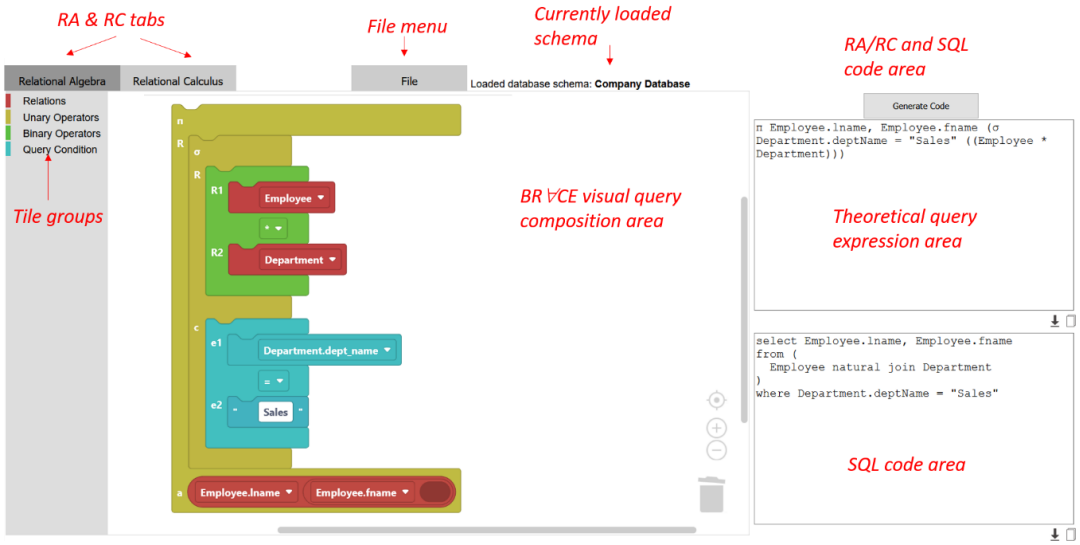


Fig. 1. The BR∀CE interface

In this paper, we present the BR∀CE interface and utilize it to formulate example RA and RC queries. The internal workings of the tool, including the developed compilers, are a subject of another publication.

The intention of this paper is to take educators through the steps of using BR∀CE for formulating RA and RC queries. That is, our objective is to provide this primer to those educators who would incorporate BR∀CE in their courses and encourage their students to utilize it. This will also provide opportunities for evaluating the tools and assess how it is helping in the teaching and learning processes.

The remainder of the paper is organized as follows. Section II discusses related work. Section III walks through the steps of using BR∀CE to formulate RA and RC queries. Section IV concludes the paper and discusses possible future research projects.

II. Related Work

Other tools that target the teaching and learning of RA and RC exist. These are listed in Table I. The table also compares these tools against each other and against BR∀CE. The comparison criteria used is whether the tool:

1) supports RA,
2) supports RC,
3) generates SQL code,
4) works with any database schema,
5) is independent (it does not require additional libraries, extra installation elements, or any other additional steps to be used),
6) is Web-based,
7) utilizes tile-based programming,
8) generates query result (directly validates the query against a specific data set, providing the data that con-stitutes the result for the query).

Supporting both RA and RC in the same tool is important since these topics are taught together in the same course. The generation of SQL code is also essential since RA and RC are simply theoretical foundations for SQL and database applications are built using SQL as a central component. The ability to work with any database schema allow the students to work with any example databases given in their courses and textbooks. Independence and Web-based access makes a tool easily available for its users. A visual, tile-based interface makes the formulation of queries easier and less prone to syntactic errors. Finally, the ability to see the query result allows the students to verify the correctness of their query. While some tools, such as ours, do not generate an immediate query results, they do, however, generate the SQL code which can be easily run against actual database implementations. That is, query verification is still possible in these tools, but requires an extra step.

DBsnap [18] allows the user to build visual queries using a tree structure. It only supports RA but does not include the division operator. It generates the RA expression, but unlike BR∀CE, it does not generate equivalent SQL to the RA. Unlike BR∀CE, it shows the user query results and intermediate results as well.

TABLE I
Tools for Teaching RA and RC

Tool/Approach	Supports RA	Supports RC	SQL Code	Any Schema	Independent	Web-Based	Tile-Based	Query Result
DBsnap [18]	✓			✓	✓		✓	✓
WinRDBI [6]	✓	✓		✓	✓			✓
iDFQL [1]	✓			✓			✓	✓
Relational [19]	✓			✓	✓			✓
RALT [15]	✓			✓			✓	✓
QVis [4]	✓			✓	✓			✓
Bags [9]	✓			✓	✓	✓	✓	
FP&P [14]	✓	✓		✓				✓
IRA [16]	✓		✓		✓			
YRA [20]	✓		✓		✓			
Relax [13]	✓			✓	✓	✓		
CalEm [7]		✓	✓		✓			
QuantX [11]		✓		✓	✓			
BR∀CE	✓	✓	✓	✓	✓	✓	✓	

WinRDBI [6] is a Windows implementation of RDBI [5]. It is a stand-alone application implemented in Prolog. The database and its schema are loaded to the tool as Prolog facts. The query is executed using the loaded database. No translation to SQL is provided and it is not clear if all the RA operations are supported.

iDFQL [1] is limited to RA. It is not maintained and is not available for testing. The tool is visual, but it does not show the RA expression. Is not independent, requiring a connection to a RDBMS to validate queries.

Relational [19] is also limited to RA. It does not generate equivalent SQL code to the RA query expression. The tool allows the user to create their own data set and schema and shows the query result.

RALT [15] requires a connection to an external Database, and it does not show the RA expression nor SQL code. The tool is not available for testing.

Query Visualiser (QVis) [4] is also limited to RA but does not support division. It executes a query and shows the results rather than generating equivalent SQL code. This tool is not available for testing either.

Bags [9] only supports RA as well. It is based on Snap [10], hence, it is Scratch-like similar to BR∀CE, but does not generate the RA expression or the equivalent SQL code.

McMaster et al. [14] describe two programming approaches. For RA, they present a function library using Visual FoxPro, and for RC, they provide a Prolog library. This is not an independent tool; instead, it is two add-ons to FoxPro and Prolog. In Table I, we call this approach FP&P. Obviously, some knowledge in FoxPro and Prolog would be required from students to use these libraries. In addition, defining the predicates that represent a database schema in Prolog adds an extra layer of overhead for students. The RA and RC queries are executed against a database without the intermediate step of generating the equivalent SQL code.

IRA [16], RA [20] (we will call it YRA using the authors initial to avoid confusion with our RA acronym), and Relax [13] are all limited to RA. RelaX and IRA provide the result of the constructed RA query. However, neither provide the user with the equivalent SQL. IRA is limited to a fixed database schema, where RelaX allows users to create their own. Both are limited by not providing support for all RA operations, namely aggregation. YRA is an RA interpreter that translates given RA queries into SQL that is then executed. YRA can provide the user with the generated SQL query, however, these are only provided through additional debugging information rather than by default for the user's benefit. YRA also must be installed directly onto the user's system rather than being an online tool, making it slightly more cumbersome to use. None of these tools provide a block-based environment to construct RA queries in.

Tools that deal with RC only also exist. We are only aware of two such tools: Calculus Emulator [7] (we used CalEm to refer to it in Table I) and QuantX [11]. CalEm is a stand-alone application that generates the equivalent SQL code. QuantX [11] is also a stand-alone application, but it simply teaches the users how to translate a complex RC query to SQL, but it does not provide any validation for the RC query.

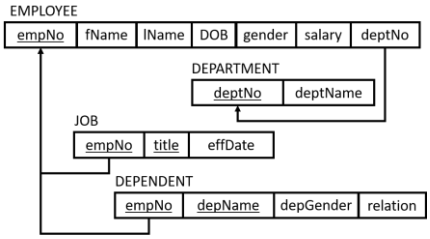


Fig. 2. A simple company database schema.

III. Composing Queries

A. Running Example

In this paper, we will use the database schema depicted in Figure 2. This database keeps information about employees, who are described by their numbers, names (first and last), date of birth (DOB), gender (we assume the gender values are M for male, F for female, X for LGPTQ, and O for other), salary, and the department (number) they work for. A department is described by its number and name. Employees may have dependents, who are described by their name, gender, and relation to the employee. The job title of an employee is also recorded with its effective start date (effDate).

B. BR∀CE Interface

The BR∀CE interface is shown in Figure 1. In BR∀CE, the schema is loaded to the tool as an XML file, which follows a very simple Document Type Definition (DTD) that lists the relations and their attribute names. To use your own schema, code in XML and load it to BR∀CE through the **File** menu choosing the *Load Database Schema* option.

Next, click the **Relational Algebra** or the **Relational Calculus** tab to start composing your RA or RC query, respectively. A query can be composed by selecting a *Tile Group* and then dragging and dropping the required tile to the *query composition area*. Some tiles require a relation or an attribute name. BR∀CE populates the field that require relation or attribute names in these tiles with drop-down lists. You can scroll down these lists and select the required relations or attributes for your query. Some tiles require a constant value or a quantifier variable name (only in RC). For these tiles, use the keyboard to enter the required value into the appropriate tile.

Once your visual query is composed and is complete, click the Generate Code button. The equivalent theoretical expression to your visual query and the equivalent SQL code will be shown in their respective areas. You can copy each of these expressions to the clipboard, using ⎘ or download them as a file, using ⬇. You can save your query from the File menu. BR∀CE queries are saved with the extensions *raq* and *rcq* for RA and RC respectively. Note that the schema representation is piggybacked to the query since queries are schema specific. When you load a query from a file, the database schema is also loaded to BR∀CE.

C. Relational Algebra Examples

We will present queries of increasing complexity, formulate these in BR∀CE, and generate the equivalent RA and SQL expressions. The operands in the RA tiles are labeled as follows:

1) R, R1, are R2 are relation operands
2) a is an attribute name list operand
3) c, c1, and c2 are logical conditions
4) e1 and e2 are expressions which can be an attribute name or a constant value entered by the user.

An operand can be left blank if the label is enclosed in square brackets, such as [a]. We do not intend, nor we have space to cover every possible RA operation supported by BR∀CE. However, we will demonstrate at least one RA operation from each tile group.

The RA tile groups are:

1) The Relations group contains the *relation* and *attribute* tiles to be used in the next two groups.

2) The Unary Operators group contains the tiles for the RA unary operators (they have one relation operand): *select*, *project*, *aggregate functions*, and *aggregate functions with grouping*.

3) The Binary Operators group contains the tiles for the RA binary operators (they require two relation operands). Many of these operators share the same tile. The required operator is chosen from a drop-down list in the tile. There are four tiles in this group: (i) joins that do not require conditions (namely, *natural join* and *cross join*), (ii) joins that require conditions (all forms of *theta joins* and *outer joins*), (iii) set operations (*intersection*, *union*, and *difference*, and (iv) *division*.

4) The Query Condition group contains the tiles for formulat-ing logic conditions. There are six tiles in this group: (i) the logical *and* or *or* tile, (ii) the logical *negation* tile, (iii) the comparison operators tile (=, ≠, <, >, ≤, and ≥), (iv) the *attribute* tile needed for attribute names in conditions, (v) the *number literal* tile, and (vi) the *string literal* tile.

Any of the tiles in the unary and Binary Operators group can serve as a container for the RA query.

1) Projection and selection: The project unary operator, denoted by the Greek symbol π, filters a relation vertically. That is, it can eliminate some of its columns. We start by formulating a projection query. Because this is our first RA query, we will go through its construction step by step in Figure 3. The query *retrieves the first name and last names of employees.*

1. Drag the *project* (π) tile from the Unary Operators group and drop it in the query composition area:

2. The *project* tile requires a relation (*R*) operand. From the Relations group, drag the *relation* tile and snap it into the *project* tile. Then, choose from the drop-down list the required relation:

3. The *project* tile also requires an attribute list operand(*a*). From the Relations group, drag the *attribute* tile and snap into the *project* tile. Then choose from the drop-down list the required attribute:

4. More attributes can be added to the attribute list by snapping more attribute tiles onto the last added *attribute* tile:

Fig. 3. Steps for creating an RA query in BR∀CE.

Once the visual query is complete, click the **Generate Code** button to generate the RA and SQL expressions that are equivalent to the BR∀CE query. The equivalent RA expression generated by BR∀CE is:

```
π Employee.fName, Employee.lName (Employee).
```

The equivalent SQL expression generated by BR∀CE is:

```
select Employee.fName, Employee.lName
from Employee.
```

The *select* operation, denoted by the Greek symbol σ, filters a relation horizontally. That is, it can eliminate some of the rows. The following is a selection query that *retrieves the employees who were born before 1970-1-1* is shown in Figure 4.

The equivalent RA expression generated by BR∀CE is:

```
σ Employee.DOB < "1970-1-1" (Employee).
```

The equivalent SQL expression generated by BR∀CE is:

```
select *
from Employee
where Employee.DOB < "1970-1-1".
```

The query in Figure 5 combines both projection and selection. It retrieves the first and last names of employees who neither identify as males nor females. The equivalent RA expression generated by BR∀CE is:

```
π Employee.fName, Employee.lName (σ
(Employee.gender ≠ "M" ∧ Employee.gender ≠ "F")
(Employee)).
```

The equivalent SQL expression generated by BR∀CE is:

```
select Employee.fName, Employee.lName
from Employee
where Employee.gender != "M"
and Employee.gender != "F".
```

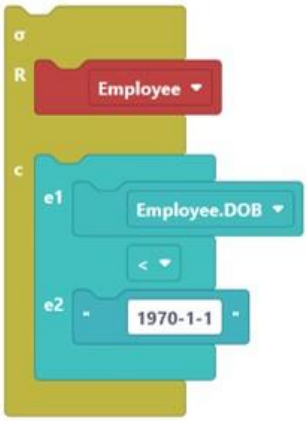


Fig. 4. A selection query in BR∀CE.

2) Aggregate functions: The aggregate functions in RA are *min*, *max*, *sum*, *count*, and *average*. Calculations can be performed to compute a single value, such as the average salary in the company, or to compute a single value for a group of rows, such as the average value per gender. The query in Figure 6 *retrieves the average salary*.

The equivalent RA expression generated by BR∀CE is:

```
AVG (Employee.salary) (Employee).
```

The equivalent SQL expression that BR∀CE generated is:

```
select AVG (Employee.salary)
from Employee.
```

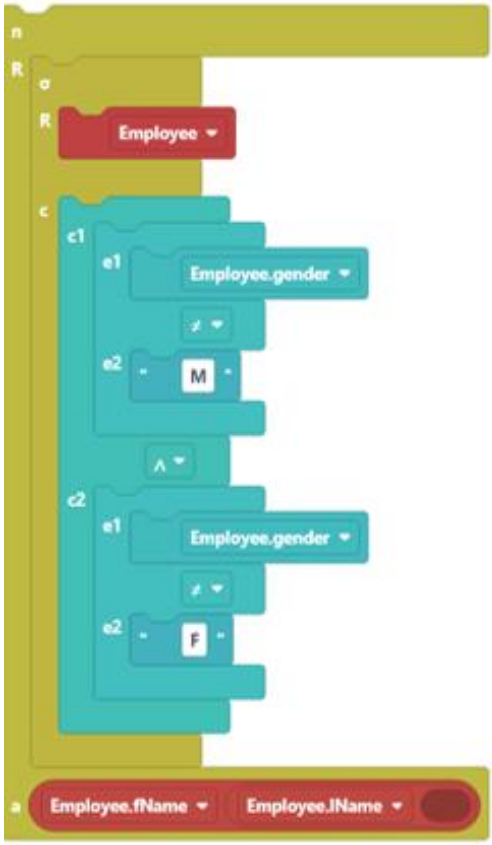


Fig. 5. A query that combines selection and projection in BR∀CE.

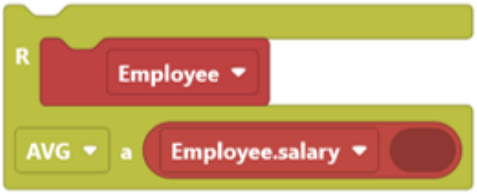


Fig. 6. An aggregate function query in BR∀CE.

To calculate any other function, such as the *min*, *max*, or *sum*, it is only necessary to choose that function from the drop-down list in the aggregate function tile. To retrieve the maximum salary for each gender group, the query is written as is shown in Figure 7.

Fig. 7. An aggregate function query with grouping in BR∀CE.

The equivalent RA expression generated by BR∀CE is:

```
MAX (Employee.salary) (Employee) (Employee.gender).
```

The equivalent SQL expression generated by BR∀CE is:

```
select Employee.gender, MAX (Employee.salary)
from Employee
group by Employee.gender.
```

3) Set operations: There are three set operations in RA: *union*, denoted by ∪, *intersection*, denoted by ∩, and *minus*, denoted by –. These are binary RA operators requiring two relations as operands. To *retrieve the depNos for departments that have male employees but do not have female employees*, we formulate the minus query in Figure 8. The first set (R1) contains departments that have male employees, and the second set (R2) contains departments that have female employees. The result is the first set minus the second set.

The equivalent RA expression generated by BR∀CE is:

```
(π Employee.deptNo (σ Employee.gender = "M" (Employee)) - π Employee.deptNo (σ Employee.gender = "F" (Employee))).
```

The equivalent SQL expression generated by BR∀CE is:

```
select Employee.deptNo
from Employee
where Employee.gender = "M"
except
select Employee.deptNo
from Employee
where Employee.gender = "F".
```

Note that the query that *retrieves the depNos for departments that have both male employees and female employees* would only require changing the "–" to "∩" in the above query. The query that *retrieves the depNos for departments that have male employees or female employees* require using "∪" instead of the "–".

4) Joins: Join queries cross reference two relations against each other by pairing each row in the first relation with each row in the second relation. For all the joins, except for the *cross join* (denoted by ×) and the *natural join* (denoted by *), a selection condition is applied to eliminate some of the irrelevant rows. The *natural join*, say of Employee and Dependent, eliminates the rows where empNo from Employee is not equal to empNo from Dependent. The * must be chosen from the drop-down list in the tile and the equivalent RA expression is: `(Employee * Dependent)`. The equivalent SQL expression generated by BR∀CE is:

```
select *
from Employee natural join Dependent.
```

This can also be expressed as an inner-join query as is shown in Figure 9. Note that the natural and inner joins eliminate the employees who do not have any dependents. To include such employees in the result with the dependent information left blank when it is not applicable (for employees who have no dependents), an *outer join* is needed. In the above query, it is sufficient to replace ⋈ by ⟕ in the tile's drop-down list to create a right (Employee) outer join. The equivalent RA expression generated by BR∀CE is:

```
(Employee ⟕ Employee.empNo = Dependent.empNo Dependent).
```

The equivalent SQL expression generated by BR∀CE is:

```
select *
from Employee right outer join Dependent on
Employee.empNo = Dependent.empNo.
```

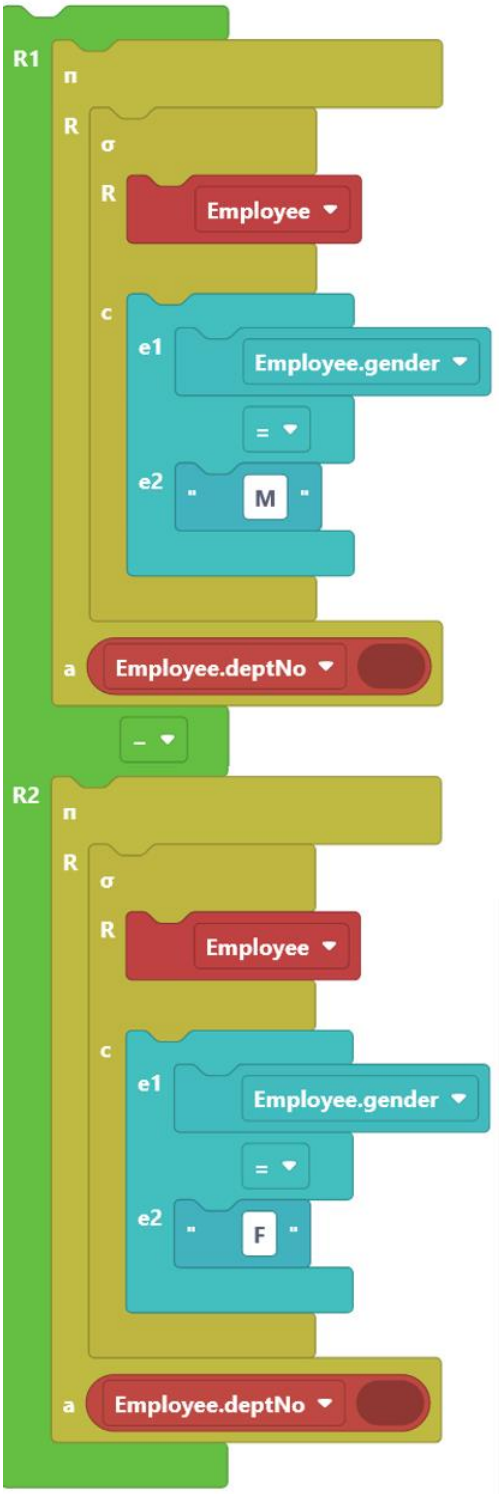


Fig. 8. A minus query BR∀CE.

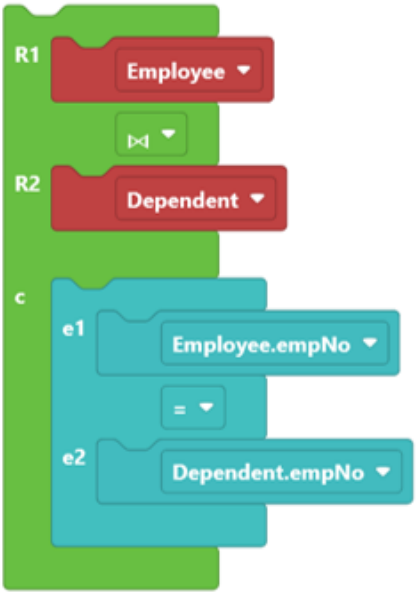


Fig. 9. A natural join query in BR∀CE.

D. *Relational Calculus Examples*

There are four groups of tiles for RC in BR∀CE:
1) The Main group contains two tiles. The *main* tile is a container for all RC queries. The *attribute* tile is used to specify attribute names in RC queries. Similarly, to RA tiles, optional operands are enclosed in square brackets.
2) The Predicates group contains two *predicate* tiles. The first tile represents the predicate *P(x)* and the second represents *P(x)*∧ *c*, where *c* is a logical condition.
3) The Quantifiers group has two tiles. The *exists* tile corresponds to the predicate ∃*x(P(x)*∧*c)* and the *forall* tile corresponds to the predicate ∀*x(P(x)* → *c)*, where *c* is a condition.
4) The Query Condition group has the same tiles as the same group in the RA tab.

1) Simple query: We start with one query that does not require the use of quantifiers. The query in Figure 10 shows the steps to formulate a query that *retrieves the employee names (first and last) who do not identify as male or female*.

The equivalent RC expression generated by BR∀CE is:
{e. fName,e.lName | Employee(e) ∧ *((e.gender* ≠ *"M")* ∧ *(e.gender* ≠ *"F"))}*.
The equivalent SQL expression generated by BR∀CE is:

```
select e.lName, e.fName
from Employee as e
where ( e.gender != "M" and e.gender != "F" ).
```

2) Existential quantifiers: Joins in RC require the use of the existential quantifier. The query in Figure 11 *lists employee names who work for the Human Resources department*.

The equivalent RC expression generated by BR∀CE is:

{e. f Name,e.lName|Employee(e) ∧ ∃ *d(Department(d)* ∧ *(d.deptName = "HumanResources")* ∧ *(d.deptNo = e.deptNo))))}*.

1. the *main* tile is required as a container:

2. This tile requires a predicate (*P*). From the Predicates group, drag and snap the simple predicate tile. Then change *Var* to *e* and choose Employee from the drop-down list:

3. Drag and snap two attribute tiles and select *fname* and *lname* from the drop-down lists:

4. Formulate the condition $(e.gender \neq \text{“}M\text{”}) \wedge (e.gender \neq \text{“}F\text{”})$:

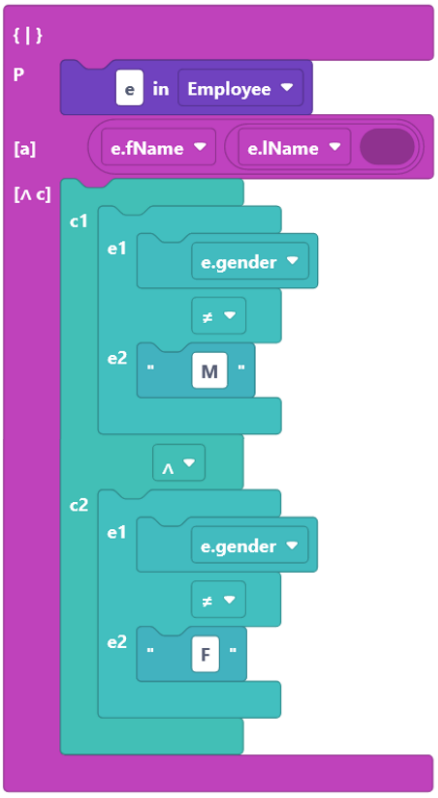


Fig. 10. Steps for creating an RC query in BR∀CE.

The equivalent SQL expression generated by BR∀CE is:

```
select e.fName, e.lName from Employee as e
where exists (
  select *
  from Department as d
  where (
          d.deptName = "Human Resources"
          and d.deptNo = e.deptNo )
).
```

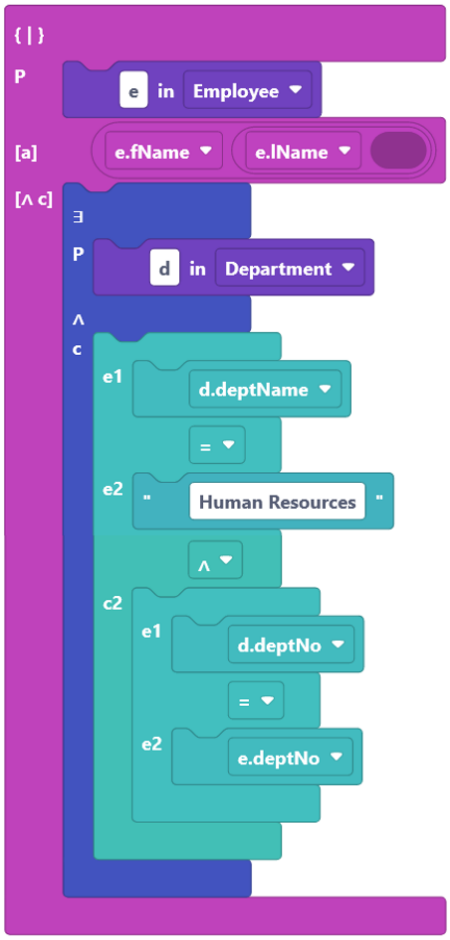


Fig. 11. An existential quantifier RC query in BR∀CE.

3) Universal quantifiers: The query in Figure 12 illustrates the use of the universal quantifier and it *retrieves the departments that have every employee earning at least 45000*. The equivalent RC expression generated by BR∀CE is:

$\{d|Department(d) \wedge \forall e((Employee(e) \wedge (e.deptNo = d.deptNo)) \rightarrow (e.salary\ \delta\ 45000))\}$.

The equivalent SQL expression generated by BR∀CE is:

```
select * from Department as d
where not (exists (
                    select *
                    from Employee as e
                    where (
                           e.deptNo = d.deptNo
                           and e.salary < 45000 )
)).
```

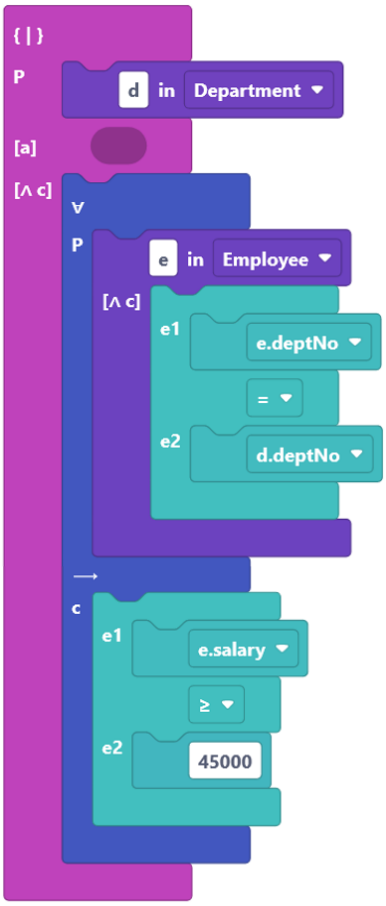


Fig. 12. A universal quantifier RC query in BR∀CE.

IV. Summary And Future Research

Using BR∀CE, RA and RC queries can be formulated by snapping tiles together. The tool allows the user to work with any relational database schema. It generates the RA or RC expression as well as equivalent SQL code. In this paper, we showed detailed examples of formulating queries that use most of the essential tiles. Previous research showed that students (and some educators) find the subject of theoretical query languages to be challenging. Given that there are many benefits for learning these languages, our freely available Web-based BR∀CE will help alleviate some of the challenges.

Several questions remain to be answered: Is there a notable effect of using BR∀CE on students learning, grades, and retention? In what ways is this tool better helping students learn the subject of theoretical query languages? How is it alleviating some of the inherent difficulties in these languages? What are the tool's limitations and how can it be improved?

The authors will be designing experiments and surveys to answer some or all these questions. The experiments will compare the performance of groups students with and without exposure to the tool. It will assess their grades in various course components that relate to query formulation. Our thesis is that the group with exposure to the tool will outperform the other group. The student surveys will aim at a minimum assessing the engagement of students with subject (will the tool improve engagement), their perception of learning (do they think the tool improved their learning.), and what improvements to BR∀CE will be needed from a functionality and usability perspectives. Now that the tool is available to the public, we also invite educators and education researchers to report on their experiences with the tool.

The introduction of tile-based programming made the subject of programming more accessible to middle-level K-12 students. We conjecture that BR∀CE will also make the subject of relational modeling and queries accessible to segments of K-12 students, specifically, junior high school students. This needs to be verified and it is our intention to do so in future research.

References

[1] A. P. Appel, E. Q. d. Silva, C. Traina Junior, and A. J. M. Traina. IDFQLl: a query-based tool to help the teaching process of the relational algebra. In *Workshop de Tecnologia da Informação no Desenvolvimento da Internet Avançada* -TIDIA. FAPESP, 2004.

[2] E. F. Codd. Relational completeness of data base sublanguages. IBM Research Report, RJ987, 1972.

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

[4] G. Constantinou. Relational algebra and SQL query visualisation. Technical Report, Department of Computing, Imperial College, UK, June 14, 2010.

[5] S. W. Dietrich. An educational tool for formal relational database query languages. *Computer Science Education*, 4(2):157–184, 1993.

[6] S. W. Dietrich, E. Eckert, and K. Piscator. Winrdbi: A windows-based relational database educational tool. In *Proceedings of the Twenty-Eighth SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '97, page 126–130, New York, NY, USA, 1997. Association for Computing Machinery.

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

[8] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Pearson, 7th edition, 2015.

[9] J. Gorman, S. Gsell, and C. Mayfeld. Learning relational algebra by snapping blocks. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, SIGCSE '14, page 73–78, New York, NY, USA, 2014. Association for Computing Machinery.

[10] B. Harvey. Bringing "no ceiling" to scratch: Can one language serve kids and computer scientists? *Constructionism*. 2010.

[11] J. Kawash. 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, 2014.

[12] J. Kawash and L. Meston. Challenges with teaching and learning theoretical query languages. In H. C. Lane, S. Zvacek, and J. Uhomoibhi, editors, *Proceedings of the 12th International Conference on Computer Supported Education*, CSEDU 2020, Prague, Czech Republic, May 2-4, 2020, Volume 2, pages 382–389. SCITEPRESS, 2020.

[13] J. Kessler, M. Tschuggnall, and G. Specht. Relax: A webbased execution and learning tool for relational algebra. In T. Grust, F. Naumann, A. Bphm, W. Lehner, T. Harder, E. Rahm, A. Heuer, M. Klettke, and H. Meyer, editors, *BTW* 2019, pages 503–506. Gesellschaft fur Informatik, Bonn, 2019.

[14] K. Mcmaster, N. Anderson, and A. Blake. Teaching relational algebra and relational calculus: A programming approach. *Information Systems Education Journal*, 01 2008.

[15] P. Mitra. Relational algebra learning tool. Technical Report, Department of Computing, Imperial College, UK, June 22, 2009.

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

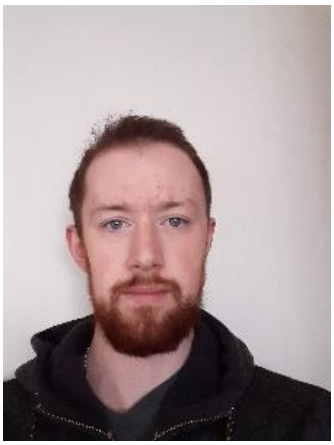
[17] A. Silberschatz, H. F. Korth, and S. Sudershan. *Database System Concepts*. McGraw-Hill, Inc., USA, 7th edition, 2019.

[18] Y. N. Silva and J. Chon. Dbsnap: Learning database queries by snapping blocks. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, SIGCSE '15, page 179–184, New York, NY, USA, 2015. Association for Computing Machinery.

[19] S. Tomaselli. Educational tool for relational algebra. https://github.com/ltworf/relational/. Accessed: 2020-09-15.

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

Jalal Kawash
received his Ph.D. in 2000 from the University of Calgary specializing in Distributed Systems. His current research interests are in Distributed Systems, Social Networks, and Scholarship of Teaching and Learning. He enjoys teaching and is the receiver of many teaching awards.



Levi Meston
finished his B.Sc. degree in Computer Science at the University of Calgary in 2020. He is currently looking to continue his education by pursuing a M.Sc. in Computer Science. He is particularly interested in pursuing studies in Computer Security.