

Relational Algebra Programming With Microsoft Access Databases

Kirby McMaster
Weber State
University,
Ogden, UT, USA

Samuel Sambasivam
Azusa Pacific
University,
Azusa, CA, USA

Nicole Anderson
Winona State
University,
Winona, MN, USA

kmcmaster@weber.edu ssambasivam@apu.edu nanderson@winona.edu

Abstract

In this paper, we describe a custom Relational Algebra Query software environment that enables database instructors to teach relational algebra programming. Instead of defining query operations using mathematical notation (the approach commonly taken in database textbooks), students write query programs as sequences of relational algebra function calls. The data to be queried can be in any Microsoft Access MDB file. The query processor software executes relational algebra instructions interactively, allowing students to view intermediate result tables. Thus, students can learn relational algebra the same way they learn SQL--through programming.

Keywords: database, query, relational algebra, programming, SQL, relational data model.

Introduction

Most commercial database systems are based on the relational data model. Recent editions of database textbooks focus primarily on the relational model. In this context, the relational model for data is arguably the most important concept in a first database course. The heart of the relational model is a set of objects called relations or tables, plus a set of operations on these objects. Coverage of the relational model in database courses includes table structures, relationships between tables, integrity constraints, and data manipulation operations (data entry and queries).

When Codd (1970) introduced his relational model of data, his main focus was on data independence. His only reference to queries mentioned predicate calculus, but not *relational algebra* (RA). Two years later, Codd (1972) gave a detailed description of relational algebra and relational calculus and considered them equal in terms of relational completeness.

Classroom discussion of database queries invariably leads to extensive coverage of SQL. Relational algebra as a query language receives less emphasis. In a survey of database educators,

Robbert and Ricardo (2003) found that only 70% included RA in their courses, compared to 92% for SQL. Nevertheless, there are significant advantages to including RA in a database course.

1. Teaching RA helps students understand the relational model. The relational model with RA operations provides a consistent, flexible way to query a database.

Material published as part of this publication, either on-line or in print, is copyrighted by the Informing Science Institute. Permission to make digital or paper copy of part or all of these works for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage AND that copies 1) bear this notice in full and 2) give the full citation on the first page. It is permissible to abstract these works so long as credit is given. To copy in all other cases or to republish or to post on a server or to redistribute to lists requires specific permission and payment of a fee. Contact Publisher@InformingScience.org to request redistribution permission.

2. Knowledge of RA facilitates teaching and learning the query portion of SQL. The basic syntax of the SQL SELECT statement provides an integrated way to combine RA operations to express a query.
3. An understanding of RA can be used to improve query performance. The query-processing component of a database engine translates SQL code into a query plan that includes RA operations. The query-optimizer attempts to speed up query execution by reducing the processing time of each operation.

When an instructor decides to include relational algebra in a database course, how should this topic be presented? RA coverage in leading database textbooks often takes a *mathematical* approach. For example, the books by Elmasri and Navathe (2006), Silberschatz, Korth, and Sudarshan (2006), and Ullman and Widom (2008) present RA concepts in mathematical terms. There are several problems with this form of representation.

Many database students are not comfortable with mathematical notation. The notation uses Greek letters in new contexts (e.g., σ for *select* and π for *project*), plus various “strange” symbols (e.g., the “bowtie” symbol--Unicode U+22C8--to represent a *join* operation). Often, several operations are combined into a single expression to make the query language appear more “algebraic”. This makes query computations hard to understand, and it disguises the procedural nature of RA. Finally, because students cannot execute a program written in mathematical notation, it is difficult to verify that the program is correct.

The mathematical approach contrasts with the way *programming* courses are taught. In a programming course, an important part of learning occurs when students write instructions for the computer and then compile and run their code. Errors in program execution provide feedback, which can reduce the gap between a student’s perception of the problem and the computer’s interpretation of the proposed solution.

All major relational database products offer SQL as the primary language for programming queries. Conversely, very few computer environments are available for developing and running relational algebra programs. Among the earliest database systems supporting RA were IS/1 (Notley, 1972) and PRTV (Todd, 1976) from IBM in England. Current systems that offer some form of RA include LEAP (Leyton, 2010) and Rel (Voorhis, 2010).

In this paper, we present a function-based relational algebra language for writing query programs and compare our syntax to that of LEAP. We then describe a custom Relational Algebra Query (RAQ) software environment that can execute query programs for Microsoft Access databases. We outline the main features of RAQ and demonstrate how to use the software to run query programs. Finally, we give several examples of RA concepts that can be taught using this approach.

Relational Algebra Programs

A relational algebra query program consists of a sequence of statements that specify operations to perform on database tables to satisfy a query. The format of the statements and the order in which they are executed vary with the RA implementation. Each statement might consist of a single RA operation, or several operations can be combined into one “algebraic” expression.

To illustrate programming of RA operations, we require an example database. The structure of a Time-and-Billing database for the fictional (television series) X-Files group within the Federal Bureau of Investigation is described next. Suppose our XFILES database consists of three tables: AGENT, CASES, and TIMECARD. The relational model for this database is shown in Figure 1. Primary keys are shown in **bold**.

This data model assumes that agents are assigned to various cases. Normally, several agents spend time on each case, and agents work on multiple cases. The number of hours an agent charges to a case each day is recorded in the TIMECARD table. Consider the following Query1 for the XFILES database.

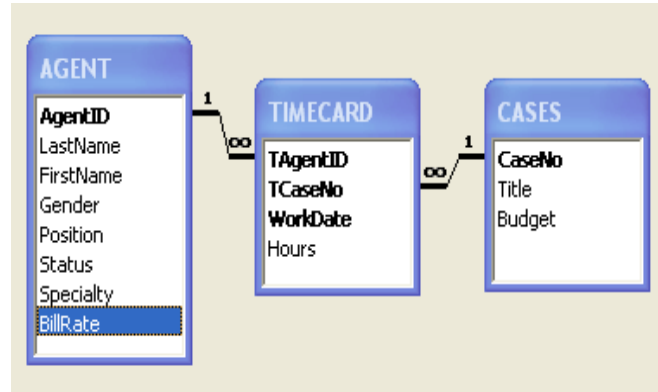


Figure 1: XFILES Database

Query1: List the agent ID and last name of all female agents that have worked on Case 2802.

We now show how Query1 can be written as a RA program in the LEAP database system. Then we will present a comparable program for this query using our function-based RA language.

Query1 for LEAP

Programming queries in LEAP using relational algebra operations is fairly straightforward. A LEAP program for Query1 is shown below.

```

R1 = join (AGENT) (TIMECARD) (AgentID=TAgentID)
R2 = select (R1) (Gender='F')
R3 = select (R2) (TCaseNo=2802)
R4 = project (R3) (AgentID, LastName)
  
```

This program performs a join of the AGENT and TIMECARD tables, followed by two row selection operations. The fourth operation projects the desired columns. In LEAP, the result of each RA operation is a new table (relation), which can be assigned a name for later use.

The above statements are written in *prefix* notation (the operation name appears before the operands), since the select, project, and join operations include conditions. RA operations without conditions--union, intersection, difference, and product--can be written in either prefix or *infix* (the operation name is placed between two operands) notation. For example, the union operation can be written in prefix form as

```
R5 = union (table1) (table2)
```

or with infix notation as

```
R5 = (table1) union (table2)
```

In LEAP, several RA operations can be combined into one nested (“algebraic”) expression. However, the LEAP documentation warns that “nesting expressions can result in rather confusing large expressions” (Leyton, 2010).

Although LEAP provides a well-designed system for performing RA queries, the data must be in a LEAP database. It cannot be in a convenient desktop database such as Microsoft Access.

Query1 for RAQ

In our relational algebra language, a query program takes the form of a sequence of function calls. Functions are defined for the nine RA operations listed in Table 1. We start each function name with the letter “T” to avoid conflicts with language keywords.

Each function receives one or two tables as arguments and returns a temporary table. The temporary table can be used in later RA operations. Using function calls for operations provides a familiar programming environment for database students.

Table 1: RA Query Functions

Operation	Function
selection	TSelect (Table1, RowCondition)
projection	TProject (Table1, ColumnList)
join	TJoin (Table1, Table2, JoinCondition)
union	TUnion (Table1, Table2)
intersection	TIntersect (Table1, Table2)
difference	TMinus (Table1, Table2)
product	TProduct (Table1, Table2)
division	TDivide (Table1, Table2)
rename	TRename (Table1, OldColumnName, NewColumnName)

Sample code for Query1 using these RA functions is shown below. In this example, each line of code performs a single operation. Our syntax is similar to the prefix notation used by LEAP.

```
-- XF Query1: XFILES Database
T1 = TJoin ('AGENT', 'TIMECARD', "AgentID=TAgentID")
T2 = TSelect (T1, "Gender='F' ")
T3 = TSelect (T2, "TCaseNo=2802")
T4 = TProject (T3, "AgentID, LastName")
```

An explanation of each line of code for this program follows.

Line 1: This is a comment (--)

Line 2: The AGENT and TIMECARD tables are *joined*. The join condition states that the AgentID (PK) field in the AGENT table must match the TAgentID (FK) field in the TIMECARD table. The output table is assigned to variable T1. Table names are placed in matching single or double quotes, since they are fixed string values. The join condition is also placed in single or double quotes.

Line 3: Rows of table T1 are then *selected* if they satisfy the condition that the Gender field value is 'F' (female). When quotation marks are needed inside a row condition, then single and double quotes should be nested in pairs. The output table variable is named T2.

Line 4: Rows of table T2 are *selected* if they satisfy the condition that the TCaseNo field equals 2802. Quotation marks are not needed for numerical values inside a row condition, since quotes are only needed for strings. The output table is assigned to variable T3.

Line 5: The two attributes of table T3 specified in the column list are *projected* as table T4, which is the final result table for the query.

Relational Algebra Software

The Relational Algebra Query software allows us to execute queries written in the format of the Query1 program. Our explanation of how to use RAQ is presented in the order the controls appear on the main screen (see Figure 2). We include sample RAQ actions that would be taken to execute Query1.

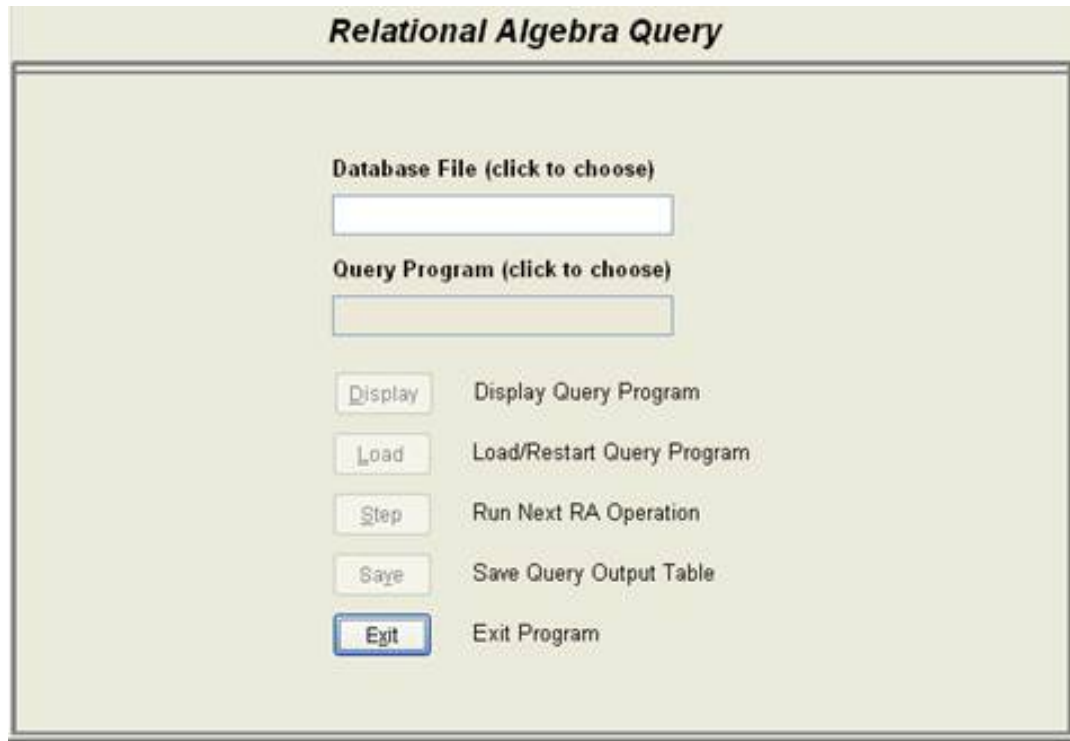


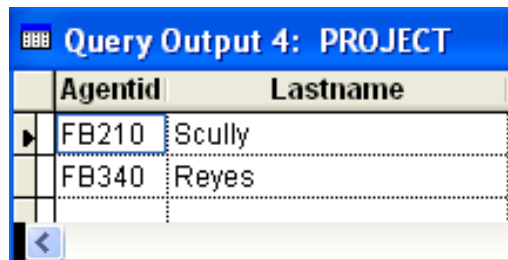
Figure 2: RAQ Main Screen

1. *Database File* textbox: Select an Access MDB file containing a database (e.g., XFILES.mdb). We chose this database format because it is ubiquitous, and MDB files are easy to distribute to students. The desired file can be picked via a file-chooser dialog box, which supports searching in subdirectories. A valid database file must be selected before other menu actions can be performed. Once selected, the database cannot be changed without exiting and then restarting RAQ.
2. *Query Program* textbox: Choose a RA query program having a TXT extension (e.g., XFquery1.txt). A new query program can be selected at any time during the execution of the RAQ software, but the following actions must be repeated.
3. *Display* button: Display the query program code in a window (optional). This action can be taken whenever the Display button is enabled. Press the Escape key to close the window. The display window is read-only. This is a handy option to refer back to the RA program during query processing.
4. *Load* button: Before a RA query program can be run, it must be loaded (initialized). This action can be repeated whenever you want to restart the program from the beginning.

5. *Step* button: Each click of this button executes one program instruction. This will normally be a single relational algebra operation. Comments in the program code are skipped. For each successful RA operation, the query output table is shown on the screen. Hit the Escape key to close the view.

If an error occurs while trying to execute an instruction, an error message is displayed in the top right-hand corner of the screen. The error message shows the code for the line that was just attempted.

Executing the XFquery1.txt program will require four steps, with four output tables (T1-T4). The initial comment line is skipped. The screen output of result table T4 using our sample XFILES database is shown in Figure 3.



Agentid	Lastname
FB210	Scully
FB340	Reyes

Figure 3: Query1 Result Table

6. *Save* button: When a RA operation has successfully completed, the current output table can be saved to disk. The format of the saved file is an Excel XLS file. The name of the output file is the name of the query program file, followed by the step number. If result table T4 is saved after step 4, the output file name will be XFquery1-4.xls.

7. *Exit* button: Click this button to exit the RAQ system. You will be prompted to confirm this request before RAQ ends. If you prefer, you can Load and rerun the same RA query program, or choose a new query program.

The availability of most RAQ menu choices depends on which actions have already occurred during program execution. Textboxes and command buttons are disabled when their selection would be inappropriate. For example, a result table cannot be saved if the current command failed to execute correctly.

Relational Algebra Concepts

The RAQ software can be used to teach relational algebra concepts interactively. The advantage of using RAQ is that students can visualize the RA concepts when they are implemented as query programs. Some examples of concepts that can benefit from this approach are described below.

Select and Join

Relational algebra is a *procedural* language, in that a sequence of operations must be specified for each query. However, several different orderings of RA operations often will produce identical solutions for a given query. Although the end result may be the same, the various code versions can differ greatly in terms of resources and performance. Consider the following query for the XFILES database.

Query2: List the agent ID and last name of all agents that have charged time to the Dark Matter case.

A RA program to find the result table for this query will involve several *selects* and *joins*. But which operations should be performed first? Generally, when the select operation precedes a join, the size of the joined table will be smaller than if the join operation is performed first. Smaller tables reduce memory requirements and should decrease processing time.

Two sample RA programs for Query2 demonstrate this concept. The Query2A program performs two joins *before* the select operation.

```
-- XF Query2A: Join before Select
T1 = TJoin('AGENT', 'TIMECARD', "AgentID=TAgentID")
T2 = TJoin(T1, 'CASES', "TCaseNo=CaseNo")
T3 = TSelect(T2, "Title='Dark Matter' ")
T4 = TProject(T3, "AgentID,LastName")
```

When the Query2A program is run, tables T1 and T2 have as many rows as the TIMECARD table (assuming referential integrity is enforced). The select operation reduces the size of table T3 to the number of rows that involve the Dark Matter case.

A second RA program for Query2 is listed below. In this version, the two joins are performed *after* the select operation.

```
-- XF Query2B: Select before Join
T1 = TSelect('CASES', "Title='Dark Matter' ")
T2 = TJoin('TIMECARD', T1, "TCaseNo=CaseNo")
T3 = TJoin('AGENT', T2, "AgentID=TAgentID")
T4 = TProject(T3, "AgentID,LastName")
```

When the Query2B program is run, table T1 contains the single row for the Dark Matter case. This will reduce the size of joined tables T2 and T3, since they contain only rows from TIMECARD that apply to the Dark Matter case. This highlights the typical advantage of joining tables “later”. By viewing the execution of these two RA programs, students can experience the size differences of intermediate tables, even though both programs lead to the same final result.

Union, Union-Compatible, and Distinct

The *union* of sets A and B consists of all distinct members of A and B. In relational algebra, the union operation requires the two input tables to be *union-compatible* (i.e., matching domains for columns). The rows in the union of two tables are *distinct*--that is, duplicate rows do not appear. This concept is illustrated by the query stated below and its accompanying Query3 program.

Query3: List the agent ID, last name, and gender of all agents that are Male or have worked on Case 2801.

```
-- XF Query3: Union Operation
T1 = TSelect('AGENT', "Gender='M' ")
T2 = TProject(T1, "AgentID,LastName,Gender")
T3 = TJoin('AGENT', 'TIMECARD', "AgentID=TAgentID")
T4 = TSelect(T3, "TCaseNo=2801")
T5 = TProject(T4, "AgentID,LastName,Gender")
T6 = TUnion(T2, T5)
```

The duplicate rows the union operation does not repeat are those in the intersection of tables T2 and T5. These rows can be viewed explicitly by adding the next line to the Query3 program.

```
T7 = TIntersect(T2, T5)
```

Intersection and Difference

Relational algebra includes three set operations: *union*, *intersection*, and *difference*. The intersection of sets A and B can be obtained from the difference operation using the following equation.

$$A \cap B = A - (A - B)$$

For example, suppose $A = \{1,2,3,4\}$ and $B = \{3,4,5\}$. Then $A - B$ is $\{1,2\}$, and $A - (A - B)$ is $\{3,4\}$. This last set is just $A \cap B$. As with union, the intersection and difference operations require the input tables to be union-compatible.

The difference operation is called Minus in Oracle and Except in the SQL standard. In our library, the function name is TMinus. The following query and its Query4 program demonstrate the RA difference operation.

Query4: List the agent ID, last name, and specialty of all agents that have *not* worked on Case 2804.

```
-- XF Query4: Difference Operation
T1 = TProject('AGENT', "AgentID,LastName,Specialty")
T2 = TJoin('AGENT', 'TIMECARD', "AgentID=TAgentID")
T3 = TSelect(T2, "TCaseNo=2804")
T4 = TProject(T3, "AgentID,LastName,Specialty")
T5 = TMinus(T1, T4)
```

If we append the following lines to the Query4 program:

```
T6 = TMinus(T1, T5)
T7 = TIntersect(T1, T4)
```

the result tables T6 and T7 will be identical.

Product and Division

The RA *division* operation is sometimes described as the “inverse” of the *product* operation, in the sense that for tables A and B,

$$(A \times B) \div B = A$$

Here, the column domains of B “match” the last column domains of $A \times B$, which is a condition required by the division operation.

Division can also be viewed as repeated intersections, where the number of intersections is not known in advance. The next query and its Query5 program provide an example of the division operation. Our RA function that performs division is called TDivide.

Query5: List the agent ID and last name of all agents that have worked on every case that Scully has worked on.

```
-- XF Query5: Division Operation
T1 = TJoin('AGENT', 'TIMECARD', "AgentID=TAgentID")
T2 = TProject(T1, "AgentID,LastName,TCaseNo")
T3 = TSelect(T1, "LastName='Scully' ")
T4 = TProject(T3, "TCaseNo")
T5 = TDivide(T2, T4)
```

Suppose we add the following line to the Query5 program:

```
T6 = TProduct(T5, T4)
```


The result table T6 will be a subset of table T2. It may not be identical to T2, because it will not include any additional “non-Scully” cases worked on by other agents. In this example, product and division behave like “almost-inverse” operations.

Limitations and Features

The RAQ software is neither feature-rich nor industrial-strength. It was designed simply to provide an academic environment for teaching relational algebra concepts through programming. Several limitations and special features of the software are described below.

1. Each RA instruction must be on a single line. By design, there is a 100-line maximum for RA query programs. This should not be a problem, since most query programs are fairly short (less than 10 lines).
2. RAQ provides modest error checking. Error messages show the offending line of code but not the reason for the error.
3. RAQ has limited input options for data. The database must be in an Access MDB (not ACCDB) file. If necessary, convert the ACCDB file to MDB format. More generally, if the database is an ODBC data source (e.g. Oracle, SQL Server, MySQL), then the table structures and data can be *imported* into an Access file.
4. RA query programs must be in a text file with a TXT extension. Programs have to be created and modified with a separate text editor, since RAQ does not provide editing capabilities.
5. RAQ output for query results are shown on the screen. The display of intermediate tables cannot be skipped, but RA programs are usually short. Query output can be saved in XLS files, and the Windows operating system provides several print-screen options.
6. For convenience in expressing queries, duplicate field names should be avoided in databases. If necessary, use the TRename function in RA programs.
7. Nesting of RA function calls within a single statement is permitted but not recommended. For example,

```
T1 = TSelect ('AGENT', "Gender='F' ")
T2 = TProject (T1, "LastName")
```

can be written in one nested statement as

```
T3 = TProject (TSelect ('AGENT', "Gender='F' "), "LastName")
```

Nested function calls defeat the opportunity to see intermediate result tables (in this case, table T1). Nesting also disguises the procedural nature of relational algebra.

8. *Date* data types can be included in queries, but there may be a need to specify date constants in row conditions. In Access, date constants are defined using the # symbol (e.g., #07/04/2010#). Few other database systems use this format for dates. One problem with the Access date format is that the position of month and day varies by country, making the above date ambiguous (July 4 or April 7?).

We have included a special *toDate* function in RAQ to allow date constants to be specified as a date datatype. Oracle has a similar *to_date* function. The format for our *toDate* function is:

```
toDate (year, month, day) .
```

Using our XFILE database as an example, suppose we want to view TIMECARD entries for the week of November 8-12, 2010. The following lines of code use the *toDate* function to specify this date range restriction:

```
T4 = TSelect('TIMECARD', "WorkDate>=toDate(2010,11,8)")
T5 = TSelect(T4, "WorkDate<=toDate(2010,11,12)")
```

9. The RAQ software has been tested in Windows XP, Windows Vista, and Windows 7. Administrative privileges may be required for Vista or Windows 7, since RAQ writes temporary files to the disk.

Summary and Conclusions

We have argued that, when teaching relational algebra to database students, a programming approach is preferable to a mathematical approach. Our favored programming style is to write query programs as sequences of function calls, where each call performs one relational algebra operation. Using this format, students can gain experience with a procedural query language while learning relational algebra.

Writing relational algebra programs improves understanding, but learning is enhanced if students can execute their query programs. We have developed a Relational Algebra Query (RAQ) software environment in which RA programs can be run. Data can be in any Access MDB file.

The RAQ software allows students to see intermediate result tables during the sequence of relational algebra operations. With this capability, students can visualize RA concepts and explore performance issues. Thus, they can learn RA in a manner similar to how they learn SQL—by writing code and watching it run.

Future Research

We have been using evolving versions of the RAQ software in database courses for several years. Most of our evidence regarding student satisfaction and the effectiveness of the software in teaching SQL has been favorable, but anecdotal. Students seem to be enjoying writing and running RA queries in the RAQ environment. However, no formal measuring instruments for the effectiveness of RAQ have been developed.

We have found that, when students have prior experience with RAQ, we change the way we teach querying using SQL. We can show students how to perform individual RA operations within the SELECT statement. Conversely, we can present complex queries and ask students what RA operations are being performed. Many subqueries can be explained in terms of RA operations. Finally, we discuss non-RA query features in SQL SELECT, including those provided by the ORDER BY clause, the GROUP BY clause, and aggregate functions.

We hope to remedy our lack of empirical measurement in the coming semesters. We plan to develop questionnaires that can assess what students like and dislike about RAQ. We also are devising questions and problems that will help us determine the effect of relational algebra programming on students' understanding of SQL. Our goal is to improve and expand the conceptual framework for our database courses through innovative programming projects.

Note: An executable version of the RAQ program, along with runtime files and the database examples in this paper, can be obtained from the lead author.

References

- Codd, E. F. (1970) A relational model of data for large shared data banks. *Communications of the ACM*, 13(6), 377-387.
- Codd, E. F. (1972). Relational completeness of data base sublanguages. In R. Rustin (Ed.), *Data base systems*. Courant Computer Science Series 6. Prentice Hall.
- Elmasri, R., & Navathe, S. (2006). *Fundamentals of database systems* (5th ed.). Addison-Wesley.

- Leyton, R. (2010). *LEAP RDBMS: An educational relational database management system*. Retrieved from: <http://leap.sourceforge.net>
- Notley, M. (1972). *The Peterlee IS/I System*. Rep. UKSC-18, IBM UK Scientific Centre, Peterlee, England.
- Robbert, M., & Ricardo, C. (2003). Trends in the evolution of the database curriculum. *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education, Greece*.
- Silberschatz, A., Korth, H., & Sudarshan, S. (2006). *Database system concepts* (5th ed.). McGraw Hill.
- Todd, S. (1976). The Peterlee Relational Test Vehicle - A system overview. *IBM Systems Journal*, 15(4), 285-308.
- Ullman, J., & Widom, J. (2008). *A first course in database systems*. Prentice Hall.
- Voorhis, D. (2010). *An implementation of Date and Darwen's Tutorial D Database Language*. Retrieved from: <http://dbappbuilder.sourceforge.net/Rel.php>

Biographies



Dr. Kirby McMaster recently retired from the Computer Science Department at Weber State University. His primary research interests are in Database Systems, Software Engineering, and Frameworks for Computer Science.



Dr. Samuel Sambasivam is chairman of the Computer Science Department at Azusa Pacific University. His research interests include Optimization Methods, Expert Systems, Client/Server Applications, Database Systems, and Genetic Algorithms.



Dr. Nicole Anderson is an Assistant Professor in the Computer Science Department at Winona State University. Her teaching and research interests involve Database Systems, Software Engineering, and the development of collaborative online learning communities.