An Object-Oriented Relational Algebra Query Language

Kirby McMaster
Fort Lewis College, Durango, CO, USA
kmcmaster@fortlewis.edu

Samuel Sambasivam
Azusa Pacific University, Azusa, CA, USA
ssambasivam@apu.edu

Brian Rague
Weber State University, Ogden, UT, USA
brague@weber.edu

Abstract

This paper describes an object-oriented query language that allows database instructors to teach Relational Algebra (RA) programming using a familiar object-oriented syntax. Instead of using mathematical notation to express queries, students write programs that instance RA table objects and call RA table methods to perform query operations. We show how writing object-oriented RA (OORA) programs can help teach RA concepts. We also present our special OORAQ query processing software that executes OORA instructions interactively, allowing students to view intermediate query results step-by-step. The database to be queried can be in any Microsoft Access MDB file. Using the OORA language and OORAQ software, students can learn Relational Algebra by writing object-oriented code and watching it run (similar to how they learn in other programming courses).

Keywords: relational database, query, relational algebra, object-oriented programming, object, method.

Introduction

Most current database systems are based on the relational data model. Recent database textbooks also focus primarily on the relational model. In this regard, the relational model for data is probably the most important concept in a relational database course.

When Codd (1970) introduced his relational model of data, he formulated it mathematically in terms of domains, n-tuples, and relations. His main focus in the original paper was on data independence, not query languages. Two years later, Codd (1972) gave a detailed definition of Relational Algebra (RA) in a collection of eight unary and binary operations (restriction, projection, product, join, division, union, intersection, and difference) that can be performed on relations.

According to Date (2004), a "relation is basically just a mathematical term for a table." From his point of view,

> A relational system is a system in which:
>
> 1. The data is perceived by the user as tables (and nothing but tables).
> 2. The operators available to the user ... are operators that derive "new" tables from "old" ones.

If the relational model were introduced today using the predominant *object-oriented* framework in Computer Science, the model would likely be defined as a set of *objects* called tables, plus a set of operations on tables (Date, 2004). The operations corresponding to Relational Algebra instructions would be implemented as *methods* in a table class.

Students enrolled in a relational database course will usually have prior exposure to object-oriented principles and languages. This should make the concept of a table class for objects and methods familiar and understandable. Consequently, this object-oriented representation would provide a pathway for students to appreciate the role of RA in the specification and implementation of database queries.

Why Teach Relational Algebra?

Course lectures on database queries invariably lead to extensive coverage of SQL. Relational Algebra as a query language receives less emphasis. In a survey of database educators, Robbert and Ricardo (2003) reported that only 70% included RA in their courses. However, there are notable benefits to including RA in a database course.

1. Teaching RA helps students better understand the relational model. The relational model with accompanying RA operations provides a well-defined procedural way to query a database.
2. Knowledge of RA helps students become more aware of what underlies the query portion of SQL. The SQL SELECT statement provides a way to combine multiple RA operations into a single query expression.
3. An understanding of RA can be used to analyze 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 the operations.

How to Teach Relational Algebra?

When an instructor includes Relational Algebra in a database course, how should RA be presented? RA coverage in many database textbooks takes a *mathematical* approach. The books by Elmasri and Navathe (2006), Silberschatz, Korth, and Sudarshan (2006), and Ullman and Widom (2008) all define RA concepts in mathematical terms. There are several problems with the mathematical represention of RA queries.

1. Most database students are uncomfortable with mathematics in general, and with the math notation used in RA queries in particular. RA math expressions in textbooks employ symbols from other languages, subscripts, and both prefix and infix operators in unfamiliar and potentially confusing ways..
2. Several operations are often combined into a single expression to make the query language appear more "algebraic". This practice disguises the order of separate query operations, and hides the procedural nature of RA.

3. Students cannot execute query programs written in mathematical notation. This makes it almost impossible for students to verify that a program is correct.

This mathematical approach does not align with the way *programming* languages are taught. In a programming course, an important part of learning occurs when students write instructions for the computer and then attempt to compile and run their code. Errors in program syntax, logic, and execution provide feedback. This can narrow the gap between a student's perception of the problem and the computer's interpretation of the proposed solution. If we write code and cannot run it, we are providing a real-life instance of a remark by Knuth (2012): "Beware of bugs in the above code; I have only proved it correct, not tried it."

All major relational database products offer SQL as the primary language for performing queries. If we desire a programming approach for teaching RA, which software systems will allow us to do so, and what syntax do these systems use to express RA queries?

Few computer environments are available for executing Relational Algebra programs. Early prototype systems supporting RA were IS/1 (Notley, 1972) and PRTV (Todd, 1976), created by IBM in England. Current systems that offer some form of RA include LEAP (Leyton, 2010), Rel (Voorhis, 2010), and WinRDBI (Dietrich, 2001).

In an earlier paper (McMaster, et. al., 2011), we proposed a function-based syntax for expressing RA queries. We described a procedural library that provided a function for each of the basic RA operations. All functions required one or two tables as input parameters (along with other parameters, as necessary). The return value of each function was a new table. A query program consisted of a sequence of function calls, until the final result table was obtained.

In this paper, we present an Object-Oriented Relational Algebra (OORA) language for writing query programs. We give several examples of Relational Algebra concepts that can be effectively taught using this object-oriented query language. We then describe an Object-Oriented Relational Algebra Query (OORAQ) software environment for executing OORA programs. We outline the main features of OORAQ, and demonstrate how to use the software to run query programs.

Object-Oriented Relational Algebra

An Object-Oriented Relational Algebra query program takes the form of a list of statements that declare database table objects and specify methods to perform on the table objects to satisfy a query. To illustrate the programming of OORA operations, we present a WEBWORK database for a consulting firm that develops web applications for clients. Our WEBWORK database consists of three tables: STAFF, JOB, and WORKLOG. The relational model for this database is displayed in Figure 1. Primary keys are shown in **bold**.

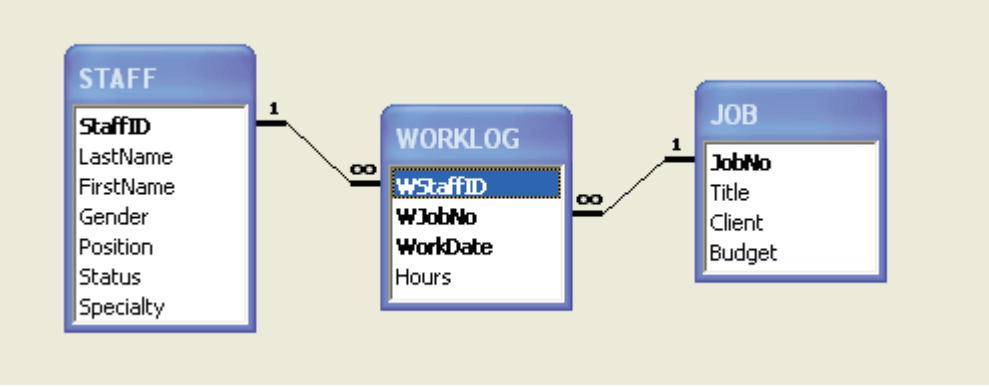

Figure 1: WEBWORK Database

This data model assumes that staff members are assigned to various jobs. Normally, each job requires several staff members, and each staff member works on multiple jobs. The number of hours a staff member charges to a job each day is recorded in the worklog. Sample data for this database are shown in the Appendix.

Object-Oriented Query Language

In our Object-Oriented Relational Algebra language, a query program includes declarations of RA table objects, followed by a sequence of method calls for objects of a RATable class. RATable methods are defined for the nine OORA operations described in Table 1. The first eight operations were introduced by Codd (1972). The ninth operation ("rename") was recommended by Date (2004).

Table 1: RATable Methods

OORA Operation	RATable Method
selection (restriction)	Table1.select(RowCondition)
projection	Table1.project(ColumnList)
join	Table1.join(Table2, JoinCondition)
union	Table1.union(Table2)
intersection	Table1.intersect(Table2)
difference	Table1.minus(Table2)
product	Table1.product(Table2)
division	Table1.divide(Table2)
rename	Table1.rename(OldColName, NewColName)

Each method receives zero or one RATable object as an argument (along with other arguments), and returns a RATable object. The returned RATable object can be used in later OORA operations. Using method calls for operations provides a familiar object-oriented programming environment for database students.

A Sample Query

Consider the following Query1 for the WEBWORK database.

Query1: List the staff ID and last name of all female ('F') staff members who have worked on Job 502.

An OORA program for Query1 using the Table 1 methods is shown below. (Note the similarity of this query language with Java and C++.) Line numbers are included to assist with the explanation that follows, but are not part of the code.

```
1: // Query 1: Select, Project, and Join
2: RATable Staff, WorkLog, T1, T2, T3, T4;
3: Staff = new RATable("STAFF");
4: WorkLog = new RATable("WORKLOG");
5: T1 = Staff.join(WorkLog,"StaffID=WStaffID");
6: T2 = T1.select("Gender='F'");
7: T3 = T2.select("WJobNo=502");
8: T4 = T3.project("StaffID,LastName");
```

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

Line 1: This is a comment (//).

Line 2: RATable objects for database tables and intermediate query tables are *declared*. RATable declarations can also be placed at the start of the lines where the objects are *instanced*.

Line 3: A RATable object Staff representing the STAFF table is created. Database table names are placed in matching single or double quotes, since they are fixed string values.

Line 4: A RATable object WorkLog representing the WORKLOG table is created.

Line 5: The Staff and WorkLog table objects are *joined*. The join condition states that the StaffID (PK) field in the Staff object must match the WStaffID (FK) field in the WorkLog object. The output is assigned to RATable object T1. The join condition (expressed as a string) is placed in single or double quotes.

Line 6: Rows of object 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 object is named T2.

Line 7: Rows of object T2 are *selected* if they satisfy the condition that the WJobNo field equals 502. Quotation marks are not needed for numerical values inside a row condition. Quotes are needed only for strings and characters. The output table object is named T3.

Line 8: The two attributes of object T3, StaffID and LastName, specified in the column list are *projected* into object T4, which is the final result table for the query.

Using our WEBWORK sample data, the Query1 result T4 is presented in Table 2.

Table 2: Query1 Result Table

StaffID	LastName
WS21	Blake
WS34	Linn

Teaching Relational Algebra Concepts with OORA

The OORA language can be used to teach Relational Algebra concepts within a familiar object-oriented programming framework. The advantage of using OORA is that students can visualize the RA concepts when they are written as query programs. Some examples of concepts that can benefit from this approach are presented below.

Select Before Join

Relational Algebra is a *procedural* language, in that a sequence of operations must be specified for each query. With OORA, the query program consists of a sequence of method calls for RATable objects. Several different orderings of method calls will often 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 required resources and observed performance.

Consider the following query for the WEBWORK database.

Query2: List the staff ID and last name of all staff members who have charged time to the Medical Informatics job.

An OORA program to find the result table for this query will involve *selects* and *joins*. 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 usually decrease processing time.

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

```
// Query2A: Join before Select
RATable Staff, WorkLog, Job, T1, T2, T3, T4;
Staff = new RATable("STAFF");
WorkLog = new RATable("WORKLOG");
Job = new RATable("JOB");
T1 = Staff.join(WorkLog,"StaffID=WStaffID");
T2 = T1.join(Job,"WJobNo=JobNo");
T3 = T2.select("Title='Medical Informatics'");
T4 = T3.project("StaffID,LastName");
```

When the Query2A program is run, tables T1 and T2 have as many rows as the WORKLOG table (assuming referential integrity is enforced). The select operation reduces the size of table T3 to the number of rows that involve the Medical Informatics job. The Query2A result table T4 based on our WEBWORK sample data is shown below.

Table 3: Query2A Result Table

StaffID	LastName
WS21	Blake
WS27	Huang

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

```
// Query2B: Select before Join
RATable Staff, WorkLog, Job, T1, T2, T3, T4;
Staff = new RATable("STAFF");
WorkLog = new RATable("WORKLOG");
```

```
Job = new RATable("JOB");
T1 = Job.select("Title='Medical Informatics'");
T2 = WorkLog.join(T1,"WJobNo=JobNo");
T3 = Staff.join(T2,"StaffID=WStaffID");
T4 = T3.project("StaffID,LastName");
```

When the Query2B program is run, table T1 contains a single row for the Medical Informatics job. This will reduce the size of joined tables T2 and T3, since they contain only rows from WORKLOG that apply to the Medical Informatics job. This highlights the typical advantage of joining tables "later".

By viewing the execution of the Query2A and Query2B OORA programs, students can observe the significant size differences of intermediate result tables, even though the two programs lead to the same final result. In this example, the final tables for Query2A and Query2B are identical.

The size of intermediate tables can sometimes be reduced by performing *project* operations before *joins*. However, this practice can be risky because the projected columns must include all attributes that will be needed in later operations. Students should be encouraged to follow careful code design practices to mitigate these risks.

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 staff ID, last name, and gender of all staff members who are male ('M') or have worked on Job 503.

An OORA program for Query3 is listed below.

```
// Query3: Union Operation
RATable Staff, WorkLog, T1, T2, T3, T4, T5, T6;
Staff = new RATable("STAFF");
WorkLog = new RATable("WORKLOG");
T1 = Staff.select("Gender='M'");
T2 = T1.project("StaffID,LastName,Gender");
T3 = Staff.join(WorkLog,"StaffID=WStaffID");
T4 = T3.select("WJobNo=503");
T5 = T4.project("StaffID,LastName,Gender");
T6 = T2.union(T5);
```

The two tables combined in the union are T2 and T5. Table T2 contains all male staff members, while table T5 includes staff members (both male and female) that worked on Job 503. The pro-

ject operations on T1 and T4 are needed to make T2 and T5 union-compatible. Using our WEBWORK sample data, the query result T6 is presented in Table 4.

Table 4: Query3 Result Table

StaffID	LastName	Gender
WS12	Samba-sivam	M
WS16	McMaster	M
WS21	Blake	F
WS27	Huang	M
WS34	Linn	F

The duplicate rows that are not repeated in table T6 are those in the intersection of tables T2 and T5 (males who worked on Job 503). These rows can be viewed separately by using the *intersect* method for tables T2 and T5.

Difference vs. Intersection

Relational algebra includes two additional set operations--*difference* and *intersection*. As with union, the difference and intersection 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 RATable class, the method name is *minus* (our method names are lower-case).

The following query and its Query4 program demonstrate the RA difference operation.

Query4: List the staff ID, last name, and specialty of all male ('M') employees that have *not* worked on Job 504.

An OORA program for Query4 is shown below.

```
// Query4: Difference Operation
RATable Staff, WorkLog, T1, T2, T3, T4, T5, T6;
Staff = new RATable("STAFF");
WorkLog = new RATable("WORKLOG");
T1 = Staff.select("Gender='M'");
T2 = T1.project("StaffID,LastName,Specialty");
T3 = Staff.join(WorkLog,"StaffID=WStaffID");
T4 = T3.select("WJobNo=504");
T5 = T4.project("StaffID,LastName,Specialty");
T6 = T2.minus(T5);
```

Table T2 includes the desired attributes for all male staff members. Table T5 consists of the same attributes for staff members who worked on Job 504. The resulting rows in table T6 (shown below) represent those male staff members who did *not* work on Job 504. With our WEBWORK sample data, the query result T6 is presented in Table 5.

Table 5: Query4 Result Table

StaffID	LastName	Specialty
WS12	Samba-sivam	Management
WS16	McMaster	OOP

The intersection of two union-compatible tables (such as T2 and T5 above) can be obtained by applying two repeated *minus* methods according to the following set equation:

```
A ∩ B = A - (A - B)
```

We can demonstrate this relationship to students by adding the following statements to the Query4 OORA program (after declaring RATable objects T7 and T8):

```
T7 = T2.minus(T6);  // T2 - (T2 - T5)
T8 = T2.intersect(T5);
```

We then compare the intersection obtained by two *minus* operations (T7) with the intersection obtained directly using the *intersect* method (T8).

Product vs. Divide

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

```
(A x B) ÷ B = A
```

Here, the column domains of B "match" the last column domains of `AxB`, which is a condition required by the division operation.

Division can be viewed as repeated intersections, where the number of intersections is not known in advance. The next query, Query5, and its OORA program provide an example of the division operation. The OORA method that performs division is called *divide*.

Query5: List the staff ID and last name of all employees that have worked on every job that Blake has worked on.

```
-- Query5: Division Operation
RATable Staff, WorkLog, T1, T2, T3, T4, T5;
Staff = new RATable("STAFF");
WorkLog = new RATable("WORKLOG");
T1 = Staff.join(WorkLog,"StaffID=WStaffID");
T2 = T1.project("StaffID,LastName,WJobNo");
T3 = T1.select("LastName='Blake'");
T4 = T3.project("WJobNo");
T5 = T2.divide(T4);
```

Table T2 contains the desired attributes StaffID and LastName (plus job number) for all staff members. Table T4 lists the job numbers of all jobs worked on by Blake (without duplicates). The division table T5 shows all staff members who worked on all of the Blake jobs (and possibly on additional jobs). From our WEBWORK sample data, the query result T5 is presented in Table 6.

Table 6: Query5 Result Table

StaffID	LastName
WS21	Blake
WS27	Huang

Suppose we add the following statement to the Query5 program (after declaring RATable object T6):

```
T6 = T5.product(T4)
```

Table T6 will be a subset of table T2. It may not be identical to T2, because it will not include any additional "non-Blake" jobs worked on by other staff members. In this example, *product* and *divide* behave like "almost-inverse" operations.

Object-Oriented Relational Algebra Software

Having an object-oriented relational algebra language allows students to write RA queries in terms of objects and methods. This activity should provide improvement in understanding how RA can express the relational model. However, learning will be limited unless students can run their query programs on sample databases.

We developed our Object-Oriented Relational Algebra Query (OORAQ) software to enable students to execute queries written in the format of the query programs listed in the previous section. Our explanation of how to use OORAQ is presented in the order the controls appear from top to bottom on the main screen (see Figure 2). We include sample OORAQ actions that would be taken to execute the Query1 program listed earlier.

1. *Database File* textbox: Select an Access MDB file containing a database (e.g. WEBWORK.mdb). The desired file can be picked via a file-chooser dialog box, which supports searching in subdirectories. OORAQ supports the MDB file format because these sample database files are easy to distribute to students.

   A valid MDB file must be selected before other menu actions can be performed. Once selected, the database cannot be changed without exiting and then restarting the OORAQ software.

2. *Query Program* textbox: Choose an OORA query program file having a TXT extension (e.g. WWquery1.txt). A new query program can be selected at any time while the OORAQ software is running. Each query program choice requires that the following actions (except *Display*) be repeated.

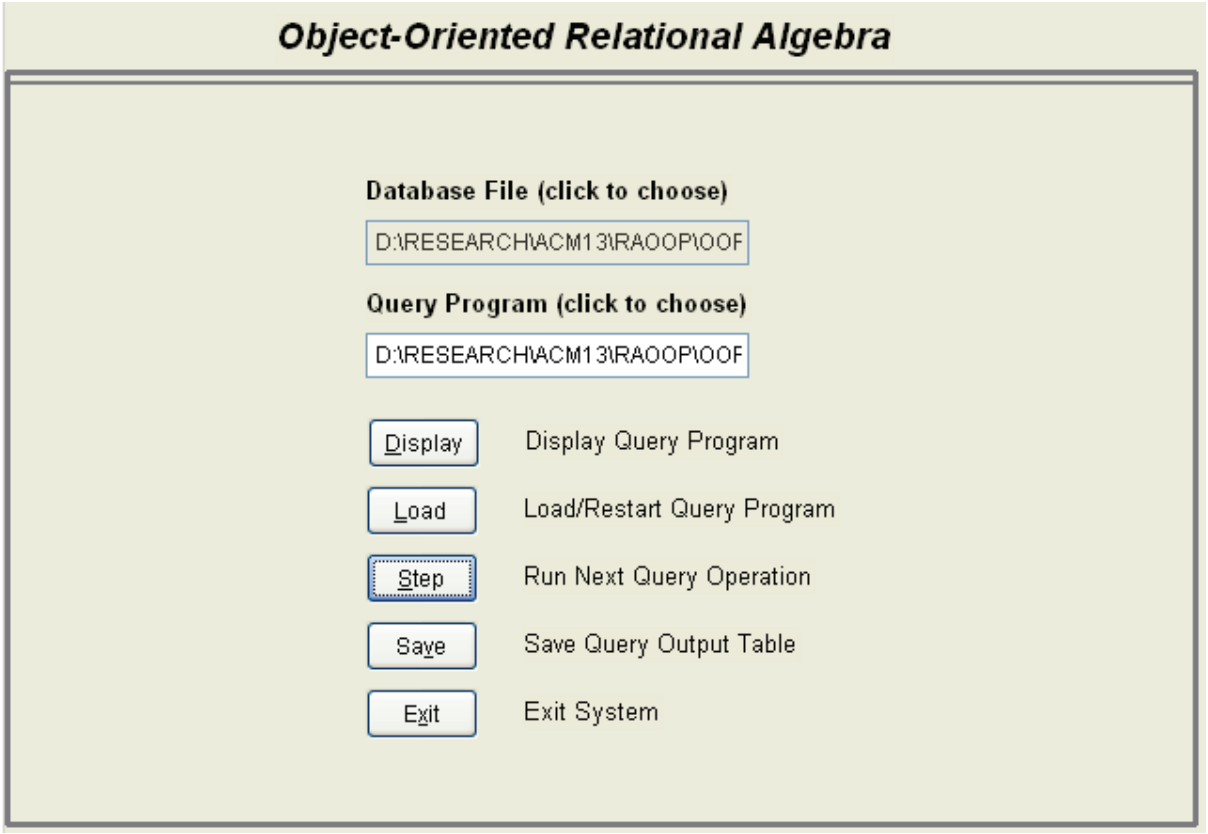

Figure 2: OORAQ Main Screen

3. *Display* button: Display the query program code in a window (optional). Press the Escape key to close the window. The display window is read-only. While executing the query program steps, it is often helpful to look again at the code that generated the intermediate results.
4. *Load* button: Before an OORA query program can be run, it must be loaded (initialized). This action must be repeated if 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 OORA operation. Comments and table object declarations in the program code are skipped. For each successful OORA operation, the output table is shown on the screen. Hit the Escape key to close the view.

   If an error occurs while executing an instruction, an error message is displayed in the top right-hand corner of the screen. The error message shows the line that was attempted. Press the Display button if you want to see this error in the context of the full query program.
6. *Save* button: When an OORA operation has successfully completed, the current output table can be saved to disk as an Excel XLS file. The name of the output file is the name of the query program file, followed by the step number.

   For example, using sample data from our WEBWORK database, the final result table T4 for Query1 in XLS format was shown earlier in Table 2. The output file would be named WWquery1-6.xls. (Comments and RATable declarations are not counted as executable steps.)
7. *Exit* button: Click this button to exit the OORAQ system. You will be prompted to confirm this request before OORAQ ends. If you prefer, you can Load and run another OORA query program.

OORAQ menu choices are enabled, based on which actions have already occurred during program execution. Textboxes and command buttons are disabled when their selection would be inappropriate.

Software Features and Limitations

The OORAQ software has been designed to provide a friendly environment for teaching Relational Algebra concepts through object-oriented programming. It is not intended to be an industrial-strength commercial product. Several features and limitations of the software are described below.

1. OORAQ has limited input options. The database must be in an Access MDB (not ACCDB) file.
2. OORA 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 OORAQ does not provide editing capability.
3. Each OORA instruction must be on a single line. There is a 100-line maximum for OORA query programs. This limit should not be an issue, since almost all OORA programs are very short.
4. OORAQ provides modest error checking. Error messages show the offending line of code but not the reason for the error.
5. Table aliases are available in SQL to distinguish between columns in different tables that have the same name. These aliases are not a feature of Relational Algebra. For this reason, duplicate field names should be avoided in sample databases. When necessary, use the *rename* method to make column names unique in OORA programs.
6. Date constants can be included in OORA query programs using the Access date format #mm/dd/yyyy# (e.g. #08/17/2014#).
7. OORAQ output for query results are shown on the screen. The display of intermediate tables cannot be skipped. Query output can be saved in XLS files.
8. The OORAQ software has been tested in Windows XP, Windows Vista, and Windows 7. Administrative privileges may be required for Vista or Windows 7.

Conclusions

We have suggested that, when teaching Relational Algebra to database students, a programming approach is preferable to a mathematical approach. We also recommended using an *object-oriented* programming syntax. Our Object-Oriented Relational Algebra (OORA) language involves writing a query program as a sequence of RATable method calls. Each OORA method performs one RA operation. Using this format, students can leverage their experience with object-oriented programming to learn RA.

Writing OORA programs improves understanding of RA, but learning is enhanced if students can execute their query programs. We have developed an Object-Oriented Relational Algebra Query (OORAQ) software environment in which OORA programs can be run. Data can be in any Access MDB file.

The OORAQ software allows students to see intermediate result tables during the executed sequence of OORA methods. With this ability, students can visualize RA concepts in action and explore query behavior. They can examine the efficiency of different RA strategies and note the

sizes of intermediate tables. In this way, they can learn RA in a manner similar to how they learn SQL: by writing and running query code and then viewing the results.

Future Research

The authors of this paper have been using evolving versions of our Relational Algebra software in database courses for several years. Most of our feedback regarding student satisfaction and the effectiveness of the software in teaching SQL has been favorable but anecdotal. Students appear to enjoy writing and running OORA queries. However, no formal study of the effectiveness of this approach has been performed.

In future research, we hope to empirically examine the relationship between writing OORA programs and learning SQL. We will attempt to measure how well a student's understanding of RA improves his/her ability to write SQL query statements, and vice versa. The main obstacle to this research is coordinating the database courses taught by the authors of this paper, since we do not teach database courses every semester.

We have observed that, when students have a good understanding of Relational Algebra, we are able to change the way we teach SQL. We can show students how to perform a sequence of OORA operations to yield the same result as a given SQL SELECT statement. Conversely, we can present complex SQL queries and ask students which RA operations are being performed. We also deal to some extent with questions of query optimization. Finally, we discuss non-RA features in the SELECT statement, such as those expressed in the ORDER BY clause, the GROUP BY clause, and aggregate functions.

Note: An executable version of the OORAQ program, along with runtime files and the sample WEBWORK database described 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. *Comm. ACM IS*, 6 (June).

Codd, E. F. (1972). Relational completeness of data base sublanguages. In Rustin & Randall (ed.), *Data base systems.* Courant Computer Science Series 6. Prentice Hall.

Date, C. J. (2004). *An introduction to database systems* (8th ed). Addison-Wesley.

Dietrich, S. W. (2001). *Understanding relational database query languages.* Prentice Hall.

Elmasri, R., & Navathe, S. (2006). *Fundamentals of database systems* (5th ed). Addison-Wesley.

Knuth, D. (2012). Donald Knuth's home page at Stanford Univ. Retrieved November 6, 2012 from http://www-cs-faculty.stanford.edu/~uno/faq.html

Leyton, R. (2010). *LEAP RDBMS: An educational relational database management system*. Retrieved November 6, 2012 from: http://leap.sourceforge.net

McMaster, K., Sambasivam, S., & Anderson, N. (2011). Relational algebra programming with Microsoft Access databases. *Interdisciplinary Journal of Information, Knowledge, and Management*, *6*, 73-83. Retrieved from http://www.ijikm.org/Volume6/IJIKMv6p073-083McMaster544.pdf

Notley, M. (1972). *The Peterlee IS/1 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).

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 on November 6, 2012 from http://dbappbuilder.sourceforge.net

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 and Professor 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. He served as a Distinguished Visiting Professor of Computer Science at the United States Air Force Academy in Colorado Springs, Colorado for a year. He has conducted extensive research, written for publications, and delivered presentations in Computer Science, data structures, and Mathematics. He is a voting member of the ACM and is a member of the Institute of Electrical and Electronics Engineers (IEEE).

Dr. Brian Rague is an Associate Professor and Chair of the Computer Science Department at Weber State University. His research interests involve parallel computing, programming languages, and opportunities for creative software engineering in the fields of education, biomedicine and physics.

Appendix: WEBWORK Database

STAFF table

StaffID	LastName	FirstName	Gender	Position	Status	Specialty
WS12	Sambasivam	Sam	M	Director	Active	Management
WS16	McMaster	Kirby	M	Consultant	Retired	OOP
WS21	Blake	Ashley	F	Senior Developer	Active	Databases
WS27	Huang	Vic	M	Developer	Active	Server-Side Design
WS34	Linn	Devon	F	Developer	Active	Client-Side Design
WS50	Linn	McKinley	F	Student Intern	Active	Networks

JOB table

JobNo	Title	Client	Budget
501	Medical Informatics	Healthy Activity Systems	21000
502	Texas Pride Newsletter	Lone Star Promotions	12000
503	Weight Loss Program	Slimmer Slowly, Inc	18000
504	Maternity Wear	Better Images & Babies	15000
505	Rugby Sports Blog	Kiwi Communications	12000
506	Vinyl Scrapbook Products	Scrapping Sisters	15000

WORKLOG table

WStaffID	WJobNo	WorkDate	Hours
WS21	501	8/07/2014	4
WS21	502	8/07/2014	4
WS27	501	8/07/2014	4
WS27	504	8/07/2014	4
WS21	503	8/08/2014	8
WS27	503	8/08/2014	8
WS21	503	8/11/2014	8
WS27	503	8/11/2014	8
WS34	502	8/11/2014	4
WS34	503	8/11/2014	4
WS21	501	8/6/2013	4
WS21	502	8/6/2013	4
WS27	502	8/6/2013	4
WS27	504	8/6/2013	4

WStaffID	WJobNo	WorkDate	Hours
WS34	502	8/12/2014	4
WS34	505	8/12/2014	4
WS21	501	8/13/2014	8
WS27	505	8/13/2014	8
WS34	505	8/13/2014	8
WS21	503	8/14/2014	8
WS27	501	8/14/2014	2
WS27	505	8/14/2014	6
WS34	505	8/14/2014	8
WS21	501	8/15/2014	2
WS21	502	8/15/2014	6
WS27	502	8/15/2014	6
WS27	504	8/15/2014	2
WS34	504	8/15/2014	4
WS34	505	8/15/2014	4