Teaching Relational Algebra and Relational Calculus: A Programming Approach

Kirby McMaster
kmcmaster@weber.edu

Nicole Anderson
nanderson1@weber.edu
Weber State University
Ogden, UT 84408

Ashley Blake
ablaketx@hotmail.com
Seabrook, TX 77586

Abstract

This paper describes how relational algebra and relational calculus can be taught using a programming approach. This is in contrast to the mathematical presentation of these topics in most database textbooks. For relational algebra, a function library implemented with Visual FoxPro allows queries to be written as a sequence of function calls--one call per relational algebra operation. For relational calculus, Prolog can be used to write non-procedural programs for queries. In each programming environment, database students experience the power and versatility of these query languages by watching their programs run. In doing so, they gain a greater understanding of the relational model and SQL.

Keywords: query language, relational algebra, relational calculus, predicate.

1. INTRODUCTION

An article by Darwen and Date entitled "On the Relational Algebra vs. Calculus" (2005) was posted last year on the dbdebunk.com web site. In this article, each author answered the question "What precisely is the difference between an algebra and a calculus...?" Their replies focused primarily on mathematical characteristics of relational algebra (RA) and relational calculus (RC), although Date did mention some database implementation issues.

RA and RC are examples of concepts that have a mathematical model and a computer implementation. In such cases, it is common for the mathematical version to differ in important ways from the computer representation. For example, a mathematical *function* is a set of ordered pairs of values, in which no two pairs can have the same first value. Each first (input) value in an ordered pair determines the second (output) value. All pairs must have both an input value and an output value. The nature of a *function* in a programming language depends on how functions are implemented in the language. With Java, some functions have no input parameters (e.g. Math.random()), and some do not have a return value (e.g. void main(String args[])).

The coverage of RA and RC in leading database textbooks usually takes a mathematical approach. For example, the texts by Connolly (2005), Elmasri (2004), and Silberschatz (2005) present these topics as mathematical concepts using mathematical (e.g. Greek letters) notation. End-of-chapter problems are "pencil-and-paper" exercises, where students write queries us-

ing the mathematical notation but are unable to run them on the computer. Many years ago Dijkstra, in his paper "On the Cruelty of Really Teaching Computer Science" (1998), suggested that students learn best by writing code and verifying it solely through formal methods. Database textbooks unwittingly support this approach in their coverage of RA and RC, since no environment is provided for students to execute RA and RC query programs.

This is in contrast to most programming courses, where an important part of learning depends on students being able to see the effects of their code as it executes. To demonstrate how a computer implementation differs from a mathematical model, students need to have software available to experiment with and observe how it behaves. Students learn mathematics and computer concepts more effectively when they can work with actual computer representations. This is particularly true when we teach students how to query using RA and RC.

Unfortunately, few computer implementations are available for RA and RC. Relational database products offer SQL as the primary query language. Some form of Query-By-Example is included with desktop database programs. One of the few database products to offer RA as a query language is the LEAP RDBMS (Leyton, 2005). Also, since RC is a form of predicate calculus, some Prolog compilers can be used to approximate RC queries.

In this paper, we will show how to write and execute RA and RC query programs. For RA queries, instead of using the LEAP RDBMS, we prefer to use a special RA function library that we have developed for the Visual FoxPro environment. For RC queries, we utilize predicate calculus features in the Turbo Prolog compiler.

2. WHY TEACH RELATIONAL ALGEBRA AND RELATIONAL CALCULUS?

The question might arise: Do we need to teach RA and RC in a database course? We often hear the statement: "Students only need to learn SQL." Robbert and Ricardo (2003) presented a paper in 2003 entitled "Trends in the Evolution of the Database Curriculum." The authors conducted surveys of database educators in 1999, 2001, and 2002, asking them which topics are included in their database courses. For the 106 respondents in the 2001 survey, 92% said they covered SQL, 70% included RA, but only 42% mentioned RC. Apparently, many database educators do not feel that RA and RC are essential topics in their courses.

We believe otherwise. Teaching SQL is a major part of a database course, but coverage of RA and RC is also important. There are several reasons for this:

1. RA provides the relational model with a flexible way to query a database. The RA operations allow specific rows and columns from a single table to be chosen to obtain the desired data. The RA operations also define how the data in separate tables can be combined both horizontally and vertically as needed for a query.
2. Knowledge of RA facilitates learning and using SQL as a query language. The basic syntax of the SQL SELECT statement provides a simple framework for combining RA operations to express a query. When writing an SQL query statement, it helps to think in terms of the RA operations needed to retrieve data from the database.
3. The query processor component of the DBMS engine translates SQL code into a query plan that includes RA operations. Anyone who writes DBMS query processing software or needs to optimize SQL query execution will benefit from an understanding of RA.
4. Codd's first database paper "A Relational Model of Data for Large Shared Data Banks" (1970) did not mention RA, but it did suggest the use of predicate calculus (the basis for RC) as a query language. RA was introduced in later papers.
5. RC is the foundation for Query-By-Example, which provides a user-friendly interface for databases.
6. Predicate calculus (and RC) can be used to develop an intelligent front-end for a database (e.g. an expert system).
7. Learning RA and RC makes a student aware of conceptual and practical differences between procedural and non-procedural query languages.

If coverage of RA and RC in a database course is considered to be important, the next issue is how to provide this coverage. The remainder of this paper describes a programming approach for teaching RA and RC.

3. TEACHING RELATIONAL ALGEBRA

RA consists of a set of unary and binary operations on tables that can be performed in sequence to yield a result table that satisfies a query. For example, suppose a database consists of two tables, STOCK and STKTYPE, which are described as follows:

STOCK	**STKTYPE**
SNo (PK)	TType (PK)
SType (FK)	TName
SName	ROP
QOH	OSize
OnOrder	

This model assumes that inventory items (stock) are divided into categories (types). Attributes that apply to individual items (e.g. QOH—quantity on hand) are recorded in the STOCK table. Attributes that apply to all items of the same type (e.g. ROP—reorder point) are included in the STKTYPE table. The two tables are linked by a common type code (STYPE and TTYPE).

> *Query*: List the stock number, stock name, quantity on hand, reorder point, and order size for all stock items in which the quantity on hand is at or below the reorder point, and no order has yet been placed (OnOrder = 'N').

The usual textbook approach for expressing queries in terms of RA operations involves using a mathematical notation that includes both functional operators (select σ, project Π) and infix operators (join $\bowtie$). The above query can be expressed in this notation as:

$$\Pi_{SNo,SName,QOH,ROP,OSize}(\sigma_{OnOrder='N'}(\sigma_{QOH<=ROP}(STOCK \bowtie_{SType=TType} STKTYPE)))$$

There are some problems with this mathematical version of RA operations:

1. Most database students are not comfortable with the mathematical notation, especially the arbitrary use of Greek letters and the "bowtie" symbol.
2. The functional notation and infix notation for operations are difficult to mix in expressions involving several RA operations. The notation also disguises the procedural nature of RA as a query language. Breaking complex expressions into several steps, each step involving one operation, can lessen these problems. For example, the query expression shown above can be divided into the following sequence of operations:

$$TEMP1 \leftarrow STOCK \bowtie_{SType=TType} STKTYPE$$
$$TEMP2 \leftarrow \sigma_{QOH<=ROP}(TEMP1)$$
$$TEMP3 \leftarrow \sigma_{OnOrder='N'}(TEMP2)$$
$$TEMP4 \leftarrow \Pi_{SNo,SName,QOH,ROP,OSize}(TEMP3)$$

3. From a programming language point of view, the notation described above is incorrect regarding the number of parameters that are involved in these operations. This applies specifically to the selection, projection, and join operations. For example, the notation

$$\sigma_{QOH<=ROP}(TEMP1)$$

   implies that the selection operation has one input parameter, a table. The subscript after the sigma symbol suggests that each row condition requires a different selection function. To implement the selection operation as a single function would require two parameters, a table and a row condition.

4. Students cannot execute query programs written in the mathematical notation. As a result, they do not receive feedback on how RA operations behave.

Instead of specifying RA queries in terms of mathematical expressions, our approach is to have a library function for each RA operation. A version of this function library is currently implemented in Visual FoxPro as a procedure file called RALGPROC. Using this approach, a query program is written as a sequence of function calls. Each library function has one or two input parameters that are tables, plus other input parameters as necessary. The output of each function is

another table. The library functions provide database students with a more familiar representation of RA operations than does the mathematical notation.

Library functions are provided for the following RA operations:

Operation	Function
selection	TSelect(Table1,RowCond)
projection	TProject(Table1,ColList)
join	TJoin(Table1,Table2,JoinCond)
union	TUnion(Table1,Table2)
intersection	TIntersect(Table1,Table2)
difference	TMinus(Table1,Table2)
product	TProduct(Table1,Table2)
division	TDivide(Table1,Table2)
rename	TRename(Table1,OldColName, NewColName)

Not all of these functions are necessary for completeness. They are included in the library because most textbooks discuss the first eight operations. The ninth operation, rename, has been recommended by Date (2004).

Note that the selection, projection, join, and rename operations include extra parameters that are not table names. For selection and join, the extra parameter is a logical expression or predicate that acts as a row condition or a join condition. For projection, the extra parameter is a list of column names. For rename, two extra parameters are needed to specify the old and new column names.

The sample query described earlier in mathematical notation can be written as a FoxPro program file, in which calls are made to functions in RALGPROC:

```
* Relational Algebra – Stock Query
set procedure to RALGPROC
T1 = TJoin('STOCK','STKTYPE',
       [SType=TType])
T2 = TSelect(T1,[QOH<=ROP])
T3 = TSelect(T2,[OnOrder='N'])
T4 = TProject(T3,
       [SNo,SName,QOH,ROP,OSize])
browse
```

Very little knowledge of FoxPro is needed to use the RALGPROC library, since the heart of each query program is a sequence of RA function calls. An *asterisk* at the start of a line indicates a comment. The *set procedure* command makes the library functions available (similar to *#include* in C). The variables T1 to T4 represent intermediate result tables (stored in memory). FoxPro string delimiters can be pairs of single quotes, pairs of double quotes, or matched pairs of square brackets. Finally, the FoxPro *browse* command displays the final result table.

More than one program variation is possible for each query, since the *procedural* nature of RA allows different sequences of operations to yield the same result table. For example, if one of the selection operations is performed before the join operation, the revised code would be:

```
* Relational Algebra – Stock Query
* Perform Select before Join
set procedure to RALGPROC
T1 = TSelect('STOCK',[OnOrder='N'])
browse
T2 = TJoin(T1,'STKTYPE',
       [SType=TType])
browse
T3 = TSelect(T2,[QOH<=ROP])
browse
T4 = TProject(T3,
       [SNo,SName,QOH,ROP,OSize])
browse
```

In the above code, the selection operation that compares QOH with ROP remains after the join operation, since the two attributes are in different tables. We have also put a *browse* statement after each function call to display the intermediate table for each RA operation as the query proceeds.

The intent in using the function library is not to determine the fastest or most efficient way to implement a query. Rather, it is to allow students to see how RA operations can be organized to extract the data requested for a query.

When using the RALGPROC library functions, problems can arise when the same column name appears in more than one table. Whenever two tables are joined, the column names in the first table are preserved, but any duplicate column names in the second table are changed. When the set operations--union, intersection, and difference--are performed, only the column names in the first table appear in the result. These problems can be avoided by using the TRename function, but students will have an easier time writing query programs if all column names in their database schema are unique.

Appendix A describes a database project that will give students practice in writing and executing RA query programs.

4. TEACHING RELATIONAL CALCULUS

In a relational database, a table or query is a special type of set--a set of rows. RC provides, through the use of predicates, a way to specify the data in tables and the result set for each query. There are two common ways to define a set mathematically: (1) list the members of the set, or (2) state properties or conditions that all members of the set must satisfy. For example, the set listed as

A = {2,3,5,7,9,11,13,17}

can also be defined as

A = {x | integer x, x > 0, x^2 < 400,
x is a prime}

All of the above conditions for x must be true for x to be a member of set A.

In programming, a *predicate* is a function in which the return value is either *true* or *false*. Each of the conditions defining the above set A can be viewed as a predicate having input parameter x. The condition "x is an integer" can be satisfied by the choice of a datatype for the parameter. The other three predicates could be written in functional form as:

positive(x)
squareLessThan400(x)
prime(x)

The predicate setA(x) could then be defined by the logical expression:

setA(x) if positive(x)
and squareLessThan400(x)
and prime(x)

Set A consists of all values of x for which this logical expression is true. This is precisely how queries are represented using RC. For the predicates defined in RC queries, however, the parameter x represents table rows or attributes.

When RC is implemented as a Prolog program, each table and query is represented by a predicate. Table predicates are defined by listing the *facts* (data). Query predicates are defined by *rules* (logical expressions). A query predicate is true whenever the logical expression for the rule is true. The Prolog query processor searches through the table predicates to determine which row and column values cause the query predicate to be true. Query output consists of all parameter values for which the query predicate is true. Because the Prolog query engine does all of the searching and pattern matching, the program needs to provide only a description of the desired data. This is why the user program is non-procedural.

A Turbo Prolog version of the sample query described in the previous section is shown below:

```
/* Inventory Database
   Stock Query (Predicate Calculus)
*/
domains
SNo,QOH,ROP,OSize = integer
SType,SName,OnOrder,TType,TName
  = symbol

predicates
STOCK(SNo,SType,SName,QOH,OnOrder)
STKTYPE(TType,TName,ROP,OSize)
QUERY1(SNo,SName,QOH,ROP,OSize)

clauses  /* Facts and Rules */
STOCK(101,"B","Prune Basket",65,"N").
/* data for other STOCK table rows
   goes here */

STKTYPE("B","Basket",60,90).
/* data for other STKTYPE table rows
   goes here */

QUERY1(SNo,SName,QOH,ROP,OSize) if
  STOCK(SNo,SType,SName,QOH,OnOrder)
  and STKTYPE(TType,TName,ROP,OSize)
  and SType = TType
  and QOH <= ROP
  and OnOrder = "N".
```

The Prolog program shown above defines *domains* for each table column using built-in integer and symbol (string) datatypes. *Predicate* headers are then defined for the STOCK and STKTYPE tables and for QUERY1. In the *clauses* section, the STOCK and STKTYPE tables are defined by facts. One sample row is shown for each table. Rows defined by facts are evaluated as true by the Prolog query engine.

The QUERY1 predicate is defined by a rule having an *if* statement with a five-part logical expression. Each part of the logical expression is treated as a predicate--two involve tables and three involve attributes. Since all parts of the expression are connected by *and*, the QUERY1 predicate is true only when all parts are true.

Prolog is a complex language with many features. Students require only a small part of Prolog to write RC queries. All they need is the ability to define predicates--one for each table using facts, and one for each query using rules.

Appendix B describes a database project that will give students practice in writing and running RC query programs.

5. SUMMARY AND CONCLUSIONS

This paper has emphasized the importance of teaching RA and RC as query languages in database courses. These languages should be taught in environments that allow query programs to be compiled and run.

Instead of using the usual mathematical notation for RA operations, we have created a function library in Visual FoxPro to perform RA queries. We have also shown how to use Turbo Prolog to express RC queries in a non-procedural form. Prolog is used because it implements the first-order predicate calculus that is the basis for RC. Sample query programs have been demonstrated in both the FoxPro and Prolog environments.

We have taught database courses for several years that use this programming approach in teaching RA and RC. Course evaluations indicate that most students enjoy having programming projects in the database course, including projects for RA and RC. We have yet to hear a student say that she/he prefers mathematical representations for these query languages.

The heart of Information Systems is software and data. Programming can provide a valuable learning tool throughout the database course, not just for SQL. Using this approach, students can learn RA and RC the same way they learn other computing concepts--by writing programs and watching them run.

6. REFERENCES

Codd, E. F., "A Relational Model of Data for Large Shared Data Banks." Communications of the ACM, June, 1970.

Connolly, Thomas and Begg, Carolyn, Database Systems: A Practical Approach to Design, Implementation, and Management (4th ed). Harlow, England: Addison-Wesley, 2005.

Darwen, Hugh, and Date, C. J., "On the Relational Algebra vs. Calculus." www.dbdebunk.com, 2005.

Date, C. J., An Introduction to Database Systems (8th ed). Addison-Wesley, Boston, MA, 2004.

Dijkstra, Edgar, "On the Cruelty of Really Teaching Computer Science." Austin, TX, 1988.

Elmasri, Ramez, and Shamkant Navathe, Fundamentals of Database Systems (4th ed). Addison-Wesley, Boston, MA, 2004.

Leyton, Richard, "LEAP RDBMS: An Educational Relational Database Management System." leap.sourceforge.net, 2005.

Robbert, Mary Ann, and Ricardo, Catherine M., "Trends in the Evolution of the Database Curriculum." ITiCSE 2003.

Silberschatz, Abraham, et al, Database System Concepts (5th ed). New York: McGraw Hill, 2005.

Appendix A: Relational Algebra Project

In this project, you will use a Visual FoxPro implementation of part of the DreamHome Rental Database in the text (Connolly, 2005). You will perform several queries on this database using a special library of *relational algebra* functions.

You will be given four FoxPro tables: BRANCH, STAFF, PROPERTY, and OWNER. Descriptions of these tables are shown below (compare with the textbook version, since field names have been revised to make them unique). Primary key and foreign key constraints are not enforced in these tables.

BRANCH (b_branchno, b_street, b_city, b_postcode)
STAFF (s_staffno, s_fname, s_lname, s_position, s_sex, s_dob, s_salary, s_branchno)
PROPERTY (p_propno, p_street, p_city, p_postcode, p_type, p_rooms, p_rent)
OWNER (o_ownerno, o_fname, o_lname, o_address, o_telno)

The data from the textbook has already been entered into the four tables.

Perform each of the following queries by writing a relational algebra program (a sequence of function calls) for each query. A special Visual FoxPro procedure file (RALGPROC.fxp) will be given to you containing the necessary relational algebra functions.

1. List the branch number, street, and city for all branch offices in the city of London.
2. List the staff number, last name, sex, and salary for all staff members that are male or earn more than 10,000 (British) pounds per year.
3. List the staff number, last name, position, and branch number for all staff members that work in the city of London.
4. List the property number, city, rooms, and rent for all rentals that have at least 3 rooms and are listed by a female staff member.
5. List the property number, city, type, and rent for all rentals that are either located in Aberdeen or listed with the branch in Glasgow.
6. List the first name and last name of staff members who work in Glasgow and of owners who own property in Glasgow.
7. List the branch number and city for all branches that do not have any male staff members.
8. List the branch number and city for all branches that have both houses and flats to rent.

Turn in printouts of the following for each of the queries:

1. The *source code* of the relational algebra program for the query.
2. The *result table* output for the query.

Appendix B: Relational Calculus Project

This project is intended to help you understand how *relational calculus* provides a non-procedural query language for relational databases. You will examine an implementation of relational calculus using *predicates* and *rules* in Turbo Prolog. The Prolog language is based on predicate calculus, which is a form of deductive logic.

In this project, you will use a Turbo Prolog definition of four tables from the DreamHome Rental Database (Connolly, 2005). You will perform eight queries on this database using Prolog's version of relational calculus.

The Turbo Prolog program files you will need can be downloaded from the Internet. The DHRENTAL.PRO text file you will be given contains slightly modified BRANCH, STAFF, OWNER, and PROPERTY tables of the DreamHome database, expressed as a Prolog program.

Write a new Prolog program that "includes" DHRENTAL.PRO. In your program you are to define a predicate for each query. Then write one or more relational calculus statements (Prolog rules) for each query. Your predicate for each query acts like a database *view*. You can then run each query as a *Goal* in the Turbo Prolog Dialog window and see the resulting output.

1. List the branch number and street for all branch offices that are either in Aberdeen or Glasgow.
2. List the staff number, last name, position, and salary for all staff members that are female and earn less than 15,000 (British) pounds per year.
3. List the last name, position, branch number, and city for all staff members that don't work in the city of London.
4. List the property number, city, rooms, rent, and staff number for all rentals that have at least 4 rooms and are listed by a female staff member.
5. List the property number, type, rent, and branch number for all rentals that are either located in London or listed with the branch in Aberdeen.
6. List the first name and last name of staff members who work in London and of owners who own property in London.
7. List the client number, first name, last name, and property number of clients who have viewed a property in Glasgow.
8. For all clients, list the client number, preferred type, maximum rent, property number, and actual rent for all properties of the client's preferred type in which the actual rent does not exceed the client's maximum rent and is also less than 400 pounds.

Turn in printouts of the following:

1. Your *Prolog program* that "includes" the DHRENTAL.PRO program and lists your Prolog statements for each query.
2. The *output* for each query.