Teaching Tip

A Simpler (and Better) SQL Approach to Relational Division

Victor M. Matos
Computer and Information Science Department
Cleveland State University
Cleveland, Ohio 44114
matos@cis.csuohio.edu

Rebecca Grasser
Information Systems Department
Lakeland Community College
Kirtland, Ohio 44094
rgrasser@acm.org

ABSTRACT

A common type of database query requires one to find all tuples of some table that are related to each and every one of the tuples of a second group. In general those queries can be solved using the relational algebra division operator. Relational division is very common and appears frequently in many queries. However, we have found that the phrasing of this operator in SQL seems to present an overwhelming challenge to novice and experienced database programmers. Furthermore, students seem to have the most problems with the SQL version commonly recommended in the database literature. We present an alternative solution that is not only more intuitive and easier to deliver in the classroom but also exhibits a better computational performance.

Keywords: Database systems, SQL, Division operator, Relational algebra, Classroom presentation, Human reactions, Code performance.

1. INTRODUCTION

Proficiency in SQL is an important skill for IS students. SQL is a relatively small and easy to use database query language. One virtue of the language is its continuous simplicity. Complex queries could be progressively decomposed into a collection of simpler SQL interrelated fragments. This structured approach works on most cases. Unfortunately, the *traditional* SQL implementation of the relational *division operator* is an exception to this observation. We have consistently found this topic to be rather troublesome for the students (and the instructor, too). Some critiques include code complexity, lack of intuitive interpretation, and departure from the simple nature of most SQL constructs. In this note, we suggest an alternative implementation of the – rather common – division operator that greatly simplifies the classroom presentation of this important database operator. In addition to clarity, the recommended solution outperforms, by many times, the traditional SQL code. We have collected empirical evidence suggesting that students find the alternate version easier to interpret and maintain.

2. THE RELATIONAL DATA MODEL AND THE DIVISION OPERATOR

The *relational data model* deals with data held into simple two-dimensional tables. Relational algebra is a compact symbolic language used to query relational databases. The basic operators of the relational algebra are the *projection*, *selection*, *Cartesian product*, *union* and *difference* (Codd 1970; Codd 1972). Those operators are the foundation for modern database query languages and have been extensively discussed in the database literature. For convenience, other useful operators were added such as different forms of *joins* (general, natural, left/right outer), *rename*, *intersection*, and *division*. The *division* operator is less common than simple *join-select-project* queries. However it is naturally applied in many common everyday queries. For instance, division could be used in solving the following problems:

(a) Find suppliers who supply *all* the red parts,
(b) Find students who have taken *all* the core courses,
(c) Find customers who have ordered *all* items from a given line of products, and so on.

The characteristic pattern of this family of inquires is the attempt to verify whether or not a candidate subject is related to *each of the values* held in a base set. That base set is called the *divisor* (or denominator T2[B]), and the table holding the subject's data is called the *dividend* (or numerator T1[A,B]). Without loosing generality, the expression *T1*[*A,B*] / *T2*[*B*] selects the A-values from the dividend table T1[A,B], whose B-values are a super-set of those B-values held in the divisor table T2[B].

2.1 An Example

Consider the tables T1[A,B] and T2[B] depicted in Figure 1. T1 represents a list of customers and the options they bought for their new cars. Column A is the customer identification number and B represents the option included in the car. For instance, customer a1 bought her vehicle with the b1, b2, and b3 options. Table T2[B] represents a particular set of options

T1	A	B
	a1	b1
	a1	b2
	a1	b3
	a2	b1
	a2	b3
	a3	b2
	a3	b3
	a3	b4
	a4	b1

T2	B
	b2
	b3

T3	A
	a1
	a3

A: Customer Number
B: Car's Option ID

T3 = T1 / T2
.

Figure 1. Customers who bought vehicles including options b2 and b3
(such as b2: *leather seats,* and b3: *winter package*).

The resulting table T3[A] identifies the customers who acquired at least those items listed in table T2[B].

3. SQL IMPLEMENTATIONS OF THE DIVISION OPERATOR

A large number of highly regarded database books (Date 1995; Desai 1990; Elmasri 1999; Kroenke 2000; O'Neil 1999; Ramakrishnan 2000; Watson 1999) describe the implementation of the division operator using the SQL syntax of Q1 (below). Even though this solution is commonly accepted in the database literature, we have found that this syntactical version is not only difficult for the programmers to understand and maintain, but also computationally complex. Instead, we propose the alternative syntactical variation called Q0.

Q0: **Alternate Version.** Computing Relational Division using membership test*, group-by*, counting, and *having* SQL constructors.

```
Q0: SELECT A
    FROM   T1
    WHERE B IN ( SELECT B FROM T2 )
    GROUP BY A
    HAVING COUNT(*) =
        ( SELECT COUNT (*) FROM T2 );
```

Version Q0 uses membership test*, group-by*, counting, and *having* SQL constructors. The "GROUP BY A" clause is responsible for splitting the rows and creating non-overlapping A-partitions. This is equivalent to separating T1[A,B] (Figure 1) according to customer. Tuples in each A-group have already been restricted by the WHERE… predicate to those whose B-value matching *any* entry in T2[B]. To continue with the example, this will select from T1[A,B] customers who have purchased either options b2 or b3. The count of tuples in each A-partition is compared with the size of table T2. In our example, the *two* rows selected from T1 need to match the two rows in T2. Only those A-groups HAVING… the same count are selected, and their A-value is finally selected.

Q1: **Classical Version**.

```
Q1: SELECT DISTINCT x.A
        FROM   T1 AS x
        WHERE  NOT EXISTS
            ( SELECT  * FROM    T2  y
                WHERE NOT EXISTS
               ( SELECT * FROM T1 AS z
                   WHERE (z.A=x.A) AND
                   (z.B=y.B)) );
```

This version is based on deeply nested sub-queries which are interconnected using doubly negated EXISTS functions. The identifiers x, y, and z are aliases of the tables T1, T2, and T1 respectively. Here the outermost SELECT statement picks a candidate x.A as a potential answer. This candidate becomes part of the final solution if there is not a tuple y in T2 (the divisor table) for which it doesn't exist a tuple z in T1 that matches the candidate's ID (x.A=z.A) but fails to match the current y value (y.B = z.B). If such y tuple exists it would create a contradiction, because there is data in T2 to which the candidate is not related to, and therefore the candidate must be rejected.

4. CODE PERFORMANCE

In (Matos 2001) an operational comparison of Q0, Q1, and other SQL versions of the division is described. That research shows that, for some samples, Q0 was between 300 to 700 times faster than Q1. The database used in (Matos 2001) is similar to that of Figure 1, and the performance estimation is controlled by the number of records in the table and the coherence between the two tables. Q0 tends to be constant or predictably linear while Q1 in general is slow and sensitive to changes of the size of the numerator table as well as the selectivity factor.

5. ZERO DIVISION

When the divisor table T2[B] is empty, the code for Q0 and Q1 produce two different results. Q0 reports an empty set, whereas Q1 enumerates each of the A-values in T1[A]. The lack of intuitive interpretation for Q1's result creates a serious philosophical problem (Date 1991). An interesting class discussion involves looking at the outputs produced by each query - where there is a zero divisor - and asking the students to interpret the meaning of the data. This discussion will show why explaining the results of an application to non-technical staff is an important skill for IS professionals.

6. HUMAN PERCEPTIONS

In a forthcoming paper, the authors provide an empirical estimation of difficulty for Q0 and Q1. In a survey conducted among graduate and undergraduate database students we have found that regardless of their academic background, experience, and practitioner's level, the experimental subjects ranked query Q1 as *more* difficult than Q0. Subjects with an Engineering or Science major, or those students with some previous database experience, were able to understand and manipulate both Q0 and Q1 with more ease than other subjects. However, less than half of the cohort was able to correctly solve query Q0 and only 30% of respondents were able to formulate the correct answer for Q1. This is a disappointing score for a group of otherwise good students.

7. CONCLUSION

The code Q1 is a classical SQL solution for relational division. However, if you combine the poor performance of Q1 to its high degree of relative difficulty, it is clear that other equivalent but improved SQL code should be used. We strongly recommend Q0, not only for its enhanced pedagogical value, but also for its better computational speed. Students need to be aware that performance could be critical in real life production environments, particularly if the computation involves large data sets. We believe the syntactical construction of Q0 allows the student to grasp the concepts of implementing SQL division in a more intuitive way.

8. BIBLIOGRAPHY

Codd, E.F., "A Relational Model of Data for Large Shared Data Banks". CACM 13, No. 6, June 1970.

Codd, E.F., "Relational Completeness of Data Base Sublanguages", In Database Systems, Courant Computer Science Symposia Series 6. Englewoods Cliffs, NJ, Prentice Hall, 1972

Date, C. J., An Introduction to Database Systems. 6th Edition, 1995. ISBN 0-201-54329-4.

Date C.J., Darwen H. "Into the Great Divide", appeared in Relational Database Writings 1989-1991, Ed. Addison-Wesley, 1992. ISBN 0-201-82459-0.

Desai, B. An Introduction to Database Systems. Ed. West Publishing CO., 1990. ISBN 0-314-66771-7.

Elmasri, R., Navathe, SR. Fundamentals of Database Systems, Third Edition. Addison-Wesley Publishing Co. 1999. ISBN 0-8053-1755-4.

Kroenke, David. Database Processing Fundamentals, Design and Implementation. Ed. Prentice-Hall, 2000. ISBN 0-13-084816-6.

Matos, V., Grasser, R., "Assessing the Performance of Various SQL Versions of the Relational Division Operator", Database Management, Auerbach Pub., Feb 2001.

O'Neil, Patrick, Database Principles, Programming, Performance. Ed. Morgan Kauffman Pub., 1999.

Ramakrishnan R., and Gehrke J., Database Management Systems 2nd Edition. Ed. McGraw-Hill, 2000. ISBN 0-07-232206-3.

Watson, Richard. Database Management – Databases and Organization. 2nd Edition. Ed. Wiley, 1999.

ISBN 0-471-18074-2.

AUTHOR BIOGRAPHIES

Victor Matos is an Associate Professor of Computer and Information Science at Cleveland State University in Cleveland, Ohio.

Rebecca Grasser is an Assistant Professor of Information Systems at Lakeland Community College in Kirtland, Ohio.