

A complete universal query operator.

Richard Elling Moe

Department of information science and media studies
University of Bergen

Abstract

We argue that the division operator in relational algebra is insufficiently equipped to handle all kinds of universal queries. This is a problem since division is widely regarded as the operational counterpart of universal quantification. Accordingly, research related to universal queries, for instance on query-optimization, is based on division. We propose generalized division operators to provide better approximations of the concept of universal quantification.

1 Introduction

In connection with databases, a universal query is a request for the retrieval of data where the specification involves quantification similar to the kind one would express using universal quantifiers in formal logic.

Given a database with facts about employees, the departments that employ them, the projects they work on and which projects are run by each department, a universal query could be, for instance:

Find the employees who work on all projects run by department 5. (1)

Universal queries are somewhat complicated compared to the typical database query. They have received a fair amount of attention, both with respect to their specification (Carlis, 1986; Codd, 1972; Dadashzadeh, 1989; Date and Darwen, 1992a, 1994; Trovåg, 2004) and the algorithms for processing them (Graefe and Cole, 1995; Rantzaeu et al., 2003).

Database query languages come in many shapes, with different approaches to handling universal queries. We shall concentrate on the relational algebra where the *division* operator is typically called into action when specifying a universal query.

Nowadays, SQL is by far the dominant query language and the relational algebra is no longer used for the practical business of querying actual databases. Nevertheless, it survives as a theoretical framework. This is partly because of its solid footing in mathematical disciplines. Also, algebra-expressions expose some structural properties which relates to procedural aspects, such as execution order. SQL, on the other hand, is flagged as being declarative. This makes the algebra more suitable for low-level specification of queries and thereby a valuable analytic tool for query-optimization techniques.

This paper was presented at the NIK-2007 conference; see <http://www.nik.no/>.

Codd (1972) introduced the division operator, which was seen as the incarnation of universal quantification in the context of relational databases. This view has persisted.

Other approaches to specifying universal queries include the use of *relational comparison*, such as the *contains* predicate which was found in early versions of SQL but abandoned in later standards (Elmasri and Navathe, 2007; Kifer, 2004). Date (2004) argues for the use of relational comparison in query languages and maintain that they offer a more convenient solution for universal queries. Modern SQL supports universal quantification for number-values by predicates such as $>ALL$. For other kinds of values there is no special operator available. Instead one relies on the logical trick of expressing \forall by means of \exists and double negation - a rather cumbersome affair in SQL.

Division has retained its special status as *the* operator for universal quantification. Specifically, work on algorithms for universal queries still focuses on implementing division (Graefe and Cole, 1995; Rantzaeu et al., 2003).

The trouble is, it has been pointed out that the division operator isn't suitable for all kinds of universal quantification (Carlis, 1986; Dadashzadeh, 1989). In section 2 we identify yet another type of universal query for which the original division operator is useless. This kind of query is easily solved in other query languages such as SQL and the relational calculus, and is even expressible in the algebra independently of division. Thus, the problem pertains to the division operator alone. Perhaps it does not quite deserve its special status, and division-based research on universal query algorithms should be revised.

Here lies the motivation behind this investigation. We aim to generalize division to handle all universal queries and thereby specify the requirements for revision of existing division-algorithms. It is beyond the present scope to perform the actual revision. Nor shall we compare division to alternative approaches to universal queries, such as relational comparison.

Basics of relational algebra

There is a wealth of literature on database theory and query languages, in which notation and terminology varies. We adopt that of Elmasri and Navathe (2007).

The relational algebra is the original query-language of the relational model for databases (Codd, 1972). For the purpose of illustration let us consider the following schema for a database containing facts about employees, departments and projects of an imaginary enterprise¹:

- $EMP(\underline{ID}, DNO)$ relates identifiers of employees to the identification-numbers of the departments that employ them.
- $PROJ(\underline{PNUM}, DNUM)$ relates each project, given by an identification-number, to the department that controls it.
- $WORK(\underline{EID}, PNO)$ relates employees and the projects they work on.

The operations of relational algebra produce relations from relations, either by means of set-operations (recall that relations are sets of tuples) or using some of the operators introduced specifically for the algebra. For instance:

¹The example is adapted from Elmasri and Navathe (2007).

- $\pi_{\text{attribute-list}}(R)$ denotes the relation obtained from the relation R by projection on to the attributes listed. For example: $\pi_{EID}(WORK)$ contains the identifiers of those who work on projects.
- $\sigma_{\varphi}(R)$ denotes the relation obtained from R by selecting the tuples that satisfy the condition φ . For example: $\sigma_{PNO=33}(WORK)$ reflects the work put in on project number 33, i.e. it contains the tuples from $WORK$ pertaining to that particular project.
- $R_1 \bowtie_{\varphi} R_2$ joins the relations R_1 and R_2 , i.e. the result contains all concatenations of R_1 - and R_2 -tuples for which the condition φ is satisfied. For example: $\pi_{EID,DNUM}(WORK \bowtie_{PNO=PNUM} PROJ)$ relates employees to the departments they work for through projects.

Special case: Joining with the condition of equal values for shared attributes is referred to as *natural join*, with shorthand notation $R_1 * R_2$. The attributes shared by R_1 and R_2 appears only once in the resulting relation.

Attributes can be renamed using ρ . For example: $\rho_{(EMP,PNO)}(WORK)$ is basically the same relation as $WORK$ except that the attribute-name EID has changed to EMP , whereas PNO remains unchanged after being renamed to itself. Furthermore, the result of a query can be assigned to a name for later reference. For example

$$\begin{aligned} EMP33 &\leftarrow \pi_{EID}(\sigma_{PNO=33}(WORK)) \\ EMP55 &\leftarrow \pi_{EID}(\sigma_{PNO=55}(WORK)) \end{aligned}$$

introduces the relations $EMP33$ and $EMP55$ containing the identifiers of employees that work on project number 33 and those that work on project number 55, respectively. Now, the query $EMP33 \cap EMP55$ would retrieve those who work for both projects.

Note that set-operators like \cap , \cup and $-$ require their argument-relations to be *union-compatible*, i.e. that they have the same number of attributes and that corresponding attributes² are associated with the same domain.

Relational completeness

In its purest form, the relational algebra comprises the selection and projection operators σ and π as well as the set operations \cup , $-$ and \times . This set of operators defines the expressive power of relational algebra³. Any set of operators which is equally expressive, or more, is said to be *relationally complete*. (A renaming-mechanism such as ρ is necessary for relational completeness, but is usually not listed explicitly among the operators.)

This way of measuring the expressive power readily lends itself to other languages. In fact, relational completeness has become a minimum requirement for relational database query languages. The relational *calculus* and the algebra are equivalent with respect to expressive power (Klug, 1982; Ullman, 1988), whereas the power of modern SQL goes beyond relational completeness (Libkin, 2003).

²Relative to some mapping of correspondence. In Elmasri and Navathe (2007) the mapping is induced by the ordering of attributes within the relation-schema. We do not go to this level of detail since such mapping may well be specified by other means and since the reader can easily induce the necessary ordering on the fly.

³We disregard extensions of the algebra designed to add to its original power, such as aggregate functions and recursive closure operators.

Other operators can be added to the algebra for practical reasons, such as the join \bowtie and division \div . Even if their introduction makes no difference when it comes to expressive power, certain types of queries becomes easier to formulate.

2 Division

Division (\div) operates on two relations where the attribute-set of the first properly includes that of the second. Given relations R and S with attribute-sets X and Y , respectively, such that $Y \subset X$; $R \div S$ denotes a relation with attribute-set $X - Y$. Specifically, $t \in R \div S$ iff $\{t\} \times S \subseteq R$.

The division operator was designed for universal queries, which otherwise are complicated to express. Take for instance the case of finding those employees who work on all projects run by department number 5. Using the division operator query (1) can be formulated as follows:

$$\begin{aligned} PRO5(PNO) &\leftarrow \pi_{PNUM}(\sigma_{DNUM=5}(PROJ)) \\ RESULT &\leftarrow WORK \div PRO5 \end{aligned} \quad (2)$$

There has been numerous reports of \div being obscure and difficult to learn (Carlis, 1986; Date and Darwen, 1994, 1992a; McCann, 2003) but more relevant for our discussion are the claims that it fails to cover the full range of universal queries.

Dadashzadeh (1989) and Carlis (1986) identify certain types of universal queries for which \div is not sufficient. They propose 'generalized division operators' to overcome the problems. However, it seems that in these cases the original division operator has been generalized beyond recognition. They apparently employ relational comparison to work around the problems. One might ask whether these should be referred to as *division-operators* at all.

A problematic query

We now look into a kind of universal query for which the standard division operator seems to be of little use. Consider the following example:

$$\textit{Find the employees who work on all projects run by their own department} \quad (3)$$

Contrast this with query (1) for those who work on the projects run by department 5. Some important differences can be pointed out. First, whereas the projects belonging to department number 5 can be extracted prior to applying division, it is impossible to secure the agreement between the departments of employees and projects by selection applied independently of a division. We elaborate this point by considering the following set:

$$\{x \mid (x,y) \in EMP \text{ and } x \in WORK \div \rho_{(PNO)}(\pi_{PNUM}(\sigma_{DNUM=y}(PROJ)))\}$$

This expression denotes the desired result of our query. Unfortunately it is not an expression of the relational algebra. However, this illustrates that query (3) corresponds to a *series* of applications of \div , each with a different 'divisor' depending on a department number y . Moreover, these divisors are based on the contents of the database, i.e. when the database state changes, so does the set of actual divisors. Hence, such a series of divisions cannot be specified once and for all. Therefore, a mechanism for adjusting the divisor internally while processing is desirable, but missing in \div .

The second point we emphasize is that with ordinary division there are only two possible roles an attribute can play: either that of being a *target*, i.e. an attribute for which values are to be retrieved and presented in the result of the division. Otherwise, the attribute is *common* to both relations and is there for the target-attributes to be measured up against. Consider the definition of $R \div S$ above and recall that $Y \subset X$. Here, $X - Y$ contain the targets, and the remaining attributes are common to R and S . In example (2) above *ESSN* is the only target and *PNO* is the only common attribute. Now, given that all attributes in a division are either targets or commons there is no room for bringing other attributes into the picture. Here lies a weakness since other attributes may also hold useful information.

In query (3) we obviously need *DNO* to modify the division but this attribute does not fit in naturally, neither as target nor common. Apparently the ordinary division operator \div is poorly equipped for handling the query.

A warning: The reader should resist temptation to work around these problems by employing some form of *correlated nested queries*, similar to those found in SQL, and attempt a solution to query (3) along the following lines:

$$\pi_{ID}(EMP \bowtie_{ID=EID} (WORK \div \pi_{PNUM}(\sigma_{DNUM=DNO}(PROJ))))$$

This is not a valid algebra expression. Observe that the σ -operation refers to the attribute *DNO* which belongs to the *EMP*-relation. That is, the occurrence of an attribute inside the division-expression is correlated with one on the outside. But this violates the traditional requirement that in a selection $\sigma_{\phi}(R)$ the attributes referred to in the condition ϕ must belong to the relation R .

This kind of correlated nested query is simply not part of the algebra repertoire and its incorporation would pertain to the entire language, not only the division-operator. Therefore we take the different approach of modifying \div itself.

But how? If we tailor-made an operator to solve the specific problems with query (3) we would still be faced with the question of whether it could handle the queries pointed out by Carlis (1986) and Dadashzadeh (1989), or indeed any other difficult universal query that might emerge. We would like a guarantee that it could handle all universal queries expressible in relationally complete languages.

There is a problem. We have found no precise definition of the concept of universal queries, only loose references to the use of \forall . Hence, we must rely on an intuitive understanding of logic to determine what counts as a universal query. Given this situation our approach is as follows: First, we define a generalized division operator that is relationally complete by itself. Such an operator can provably handle all universal queries simply because it can handle every query there is. Then, realizing that such an operator can be too general for our purposes, we consider how to relax its expressive power.

3 General Division

In this section we define a new division operator, generalizing \div by incorporating extra parameters. We first present the formal definition, followed by an explanation of the new parameters, before demonstrating their use with an example.

Definition 3.1 *Let R and S be relations with attribute-sets X and Y , respectively, such that: $C \subseteq X \cap Y$, $T \subseteq X$, $C \cap T = \emptyset$, $L \subseteq (X \cup Y) - T$. Let T_1, \dots, T_n be a partition of T into union compatible components and let ϕ and ψ be boolean conditions over $(X \cup Y) - (X \cap Y)$ (of the kind used in σ -selection.).*

General division of R by S , denoted $R \frac{T_1, \dots, T_n + L : \varphi}{C : \psi} S$, is equivalent to GD given by

$$\begin{aligned} Q &\leftarrow \pi_T(R) - \pi_T(\pi_{TUC}(\sigma_\psi(\pi_{X-Y}(R) \times S)) - \pi_{TUC}(R)) \\ Q' &\leftarrow \pi_{TUL}(\sigma_\varphi(\pi_{X-Y}(Q * R) \times S)) \\ GD &\leftarrow \pi_{T_1UL}(Q') \cup \dots \cup \pi_{T_nUL}(Q') \end{aligned}$$

T are the target-attributes, while C is the selection of common attributes the targets will be measured up against. The *retrieved tuples* are the T -values that appear in R in combination with all C -values for which the condition ψ is satisfied.

The set of retrieved tuples may be modified before presented as the result of the division: If the condition φ is specified then the retrieved tuples are filtered by excluding those that fails to satisfy φ . When the L -parameter is specified, in the form of a list of attributes, retrieved tuples are expanded with the corresponding values. A retrieved tuple may then be duplicated for combination with several such L -values. If a partition T_1, \dots, T_n of the targets is specified, the retrieved tuples are split up in their T_i -components. Each giving rise to a separate tuple in the result, possibly duplicated for combination with L -values.

Example

Imagine an enterprise similar to the one presented in section 1 but with the difference that work on projects are performed by teams, each of which consists of a pair of employees⁴. An employee may be member of several teams. Teams, rather than individual employees, are associated with a department. Each team is also registered with one or more particular skills mastered by both team-members. Finally, an attribute SEX is added to distinguish between male and female employees. A state for this database could be as follows:

<i>EMP</i>	
<i>ID</i>	<i>SEX</i>
<i>Jack</i>	<i>male</i>
<i>Jill</i>	<i>female</i>
<i>Pam</i>	<i>female</i>
<i>Tom</i>	<i>male</i>

<i>TEAM</i>		
<i>E1</i>	<i>E2</i>	<i>DNO</i>
<i>Jack</i>	<i>Jill</i>	2
<i>Jill</i>	<i>Pam</i>	1
<i>Tom</i>	<i>Jack</i>	1
<i>Tom</i>	<i>Pam</i>	2
<i>Pam</i>	<i>Jack</i>	1

<i>PROJ</i>	
<i>PNUM</i>	<i>DNUM</i>
11	1
22	2
33	2

<i>TEAMSKILL</i>		
<i>E1</i>	<i>E2</i>	<i>SKILL</i>
<i>Jack</i>	<i>Jill</i>	<i>Java</i>
<i>Jack</i>	<i>Jill</i>	<i>SQL</i>
<i>Jill</i>	<i>Pam</i>	<i>C</i>
<i>Tom</i>	<i>Jack</i>	<i>Lisp</i>
<i>Tom</i>	<i>Pam</i>	<i>Prolog</i>
<i>Pam</i>	<i>Jack</i>	<i>Perl</i>

<i>TEAMWORK</i>		
<i>E1</i>	<i>E2</i>	<i>PNO</i>
<i>Jack</i>	<i>Jill</i>	22
<i>Jack</i>	<i>Jill</i>	33
<i>Jack</i>	<i>Jill</i>	11
<i>Jill</i>	<i>Pam</i>	33
<i>Tom</i>	<i>Jack</i>	11
<i>Tom</i>	<i>Pam</i>	22
<i>Pam</i>	<i>Jack</i>	11

Now consider the universal query

*Find members, and their skills, of male+female teams that
work on all projects run by their own department.* (4)

⁴This is not too far fetched: consider the practise of pair-programming in agile software development (Tessem et al., 2005).

By a suitable join-up we can produce an extended version of *TEAMWORK* where the departments and skills of the teams as well as the gender of team-members are added:

$$TEAMWORK'(E1, E2, PNO, DNO, SKILL, SEX1, SEX2)$$

Then the query

$$TEAMWORK' \frac{E1; E2 + SKILL : SEX1 \neq SEX2}{PNO : DNO = DNUM} \rho_{(PNO, DNUM)}(PROJ)$$

will produce the desired result:

<i>E1; E2</i>	<i>SKILL</i>
<i>Jack</i>	<i>Java</i>
<i>Jack</i>	<i>SQL</i>
<i>Jack</i>	<i>Perl</i>
<i>Jill</i>	<i>Java</i>
<i>Jill</i>	<i>SQL</i>
<i>Pam</i>	<i>Perl</i>

Note how the correlation between team- and project-departments is handled using the ψ -parameter. This is the crucial difficulty for \div with query (3).

Expressive power

General division is sufficient, on its own, to express any query within the range of relational completeness.

Theorem 3.1 *General division is relationally complete.*

Proof In the following we consider relations R and S with attribute sets X and Y . Assume $X = \{x_1, \dots, x_n\}$ and $Y = \{y_1, \dots, y_m\}$ and let $X' = \{x'_1, \dots, x'_n\}$ and $Y' = \{y'_1, \dots, y'_m\}$ be sets of fresh attribute names.

Given that the set $\{\sigma, \pi, \cup, -, \times\}$ is relationally complete, it will suffice to show that any query expressed with these operators can be transformed into an equivalent query using general division only. The transformation may use the following rules:

$$\sigma_{\emptyset}(R) = R \times \rho_{(X')}(R) \frac{X+\emptyset:\emptyset}{X':false} \rho_{(X')}(R) \quad (5)$$

$$\pi_Z(R) = R \times \rho_{(X')}(R) \frac{Z+\emptyset:true}{X':true} \rho_{(X')}(R) \quad (6)$$

$$R \times S = R \frac{X+Y:true}{\emptyset:false} S \quad (7)$$

$$R \cup S = R \times \rho_{(Y')}(S) \times \rho_{(X')}(R) \frac{X;Y'+\emptyset:true}{X':true} \rho_{(X')}(R) \quad (8)$$

$$R - S = \sigma_{\bigvee_i x_i \neq y'_i} (R \times \rho_{(Y')}(S) \frac{X+\emptyset:true}{Y':true} \rho_{(Y')}(S)) \quad (9)$$

In (7) X and Y are assumed to be disjoint. In (8) and (9) R and S are assumed to be union-compatible.

These rules are applied to an expression in order to eliminate certain operators by replacing them with others. Note that rule (9) eliminates a $-$ at the cost of introducing a σ . Therefore, all occurrences of $-$ should be eliminated before applying rule (5) to eliminate occurrences of σ . Similarly, rule (7) should be applied last since all the other rules introduce occurrences of \times . Hence, if the transformation is carried out as follows

- 1 Replace all occurrences of $-$ using (9).
- 2 Replace all occurrences of σ using (5).
- 3 Replace all occurrences of π using (6).
- 4 Replace all occurrences of \cup using (8).
- 5 Replace all occurrences of \times using (7).

the result will be an expression which involves general division only. ■

On the other hand, general division is defined entirely in terms of the original operators and, hence, does not possess expressive power beyond relational completeness.

Redistribution of power

The example above with query (4) demonstrates the usefulness of general division for rather complex universal queries, but there is a disturbing circumstance. Lots of queries are not universal by nature. Intuitively they should not be handled by division if we are to maintain the view that division corresponds to the concept of universal quantification. General division, however, can handle any query and appears therefore to be too powerful.

In an attempt to remedy the situation we 'factor out' operators that correspond to other kinds of query. That is, we try to redistribute the expressive power over a selection of 'natural' operators while reducing the power of the new division correspondingly.

Which operators would qualify as 'natural' is probably a matter of debate, but one would reasonably expect the selection to be relationally complete. Pending such a debate we base our discussion on $\{\sigma, \pi, \cup, -, \times\}$ which was the original set shown to be complete, as well as minimal in the sense that none of its proper subsets are relationally complete.

Consider $\frac{T_1; \dots; T_n+L; \phi}{C; \psi}$ as in definition 3.1. The rules (5) - (9) provide clues as to how operators can be factored out. Starting with \times , we observe by rule (7) that it more or less corresponds to the '+'. Since it is not used in any other rule we conclude that the '+'-mechanism can be dropped in exchange for \times . Furthermore, since the condition *true* amounts to no condition at all, the fact that (5) is the only rule to make actual use of the condition-parameter ϕ allows us to swap it for the σ -operator. Similarly, by rule (8), \cup can replace the ';' -mechanism for partitioning. That is, we allow no other partition of T than T itself and hence the requirement for union-compatible partitions becomes obsolete.

By now we have reduced $\frac{T_1; \dots; T_n+L; \phi}{C; \psi}$ to $\frac{T}{C; \psi}$, while $-$ and π are still to be considered. First we observe that \div can simulate $-$:

$$R - S = (\sigma_{\bigvee_i x_i \neq y_i} (R \times \rho_{(Y')}(S))) \div \rho_{(Y')}(S) \quad (10)$$

So there is no reason to factor out $-$ since it was expressible by division all along.

As for π , its simulation (6) hinges on the use of T but since other rules also use this parameter it seems that T is not solely the part of a projection mechanism. Furthermore, for reasons explained below, we wish to keep T as a parameter to division.

At this point we stop the process, which leaves us with the $\frac{T}{C; \psi}$ division operator. Like T , C is used in different ways by several of the simulation rules (5) - (9) and should

therefore not be dropped as the result of factoring out any single operator. We keep the condition-parameter ψ since it allows us to refine divisors internally while processing, which is a decisive feature in enabling a division solution to query (3).

We have basically completed the re-factoring of general division, what remains is a little bit of refurbishing.

Final touches to a new division

Recall the discussion in section 2: attributes in \div -expressions are either targets or common, which makes it impossible to bring other attributes into the picture. Our new operator leaves room for such *contextual* attributes which can be used to modify the division. In definition 3.1, C specifies only some of the attributes shared by relations R and S . Likewise T are only some of the R -attributes not shared with S . The remaining attributes are contextual.

Having the option to keep common attributes outside C can save us from some simple book-keeping operations. But in order to avoid ambiguity the conditions ϕ and ψ cannot refer to attributes shared by R and S . Hence, common attributes must be renamed before they can serve as fully fledged contextuials. We also observe that in the simulations (5) - (9), C contains all common attributes. All in all we find no strong reason to allow attributes to be both common and contextual. Therefore we simplify $\frac{T}{C:\psi}$ further by requiring that C should correspond to all common attributes. That is, the targets are measured up against every common attribute, just as in \div , at the expense of some extra projection and renaming now and then. On the other hand, it is no longer necessary to list the members of C . By letting the list of targets T be a parameter to the operator, the roles of attributes are fully specified: the targets are given explicitly, the common attributes can be identified from the relation-schemas and the remaining attributes are contextual.

By now, general division has been significantly trimmed down. Let us spell out the details of the resulting division operator.

Definition 3.2 *Let R and S be relation-schemas with attribute sets X and Y respectively. Let $T \subseteq X$ and $C = X \cap Y$ such that $C \cap T = \emptyset$. Let ψ be a boolean condition over $(X \cup Y) - C$. We define $R \frac{T}{\psi} S$ to be equivalent with*

$$\pi_T(R) - \pi_T(\pi_{T \cup C}(\sigma_\psi(\pi_{X-C}(R) \times S)) - \pi_{T \cup C}(R))$$

Here, T corresponds to the targets, while C contain the common attributes. The remaining attributes are contextual. For the intended use, T and C will be non-empty.

The new operator is indeed a generalization of the original: If no contextual attributes are present and the condition evaluates to true in all cases, then $\frac{T}{\psi}$ boils down to being the same as \div .

Then, by substituting $\frac{X}{true}$ for \div in (10) we observe that $\frac{T}{\psi}$ can simulate $-$. It can also simulate π :

$$\pi_Z(R) = R \times \rho_{(X')}(R) \frac{Z}{true} \rho_{(X')}(R)$$

This implies that $\{\frac{T}{\psi}, \sigma, \times, \cup\}$ is relationally complete. Hence, factoring out σ , \times and \cup from general division has not caused a loss of overall expressive power. In this respect we can be sure that no more than these operators have been factored out.

Examples

Using $\frac{T}{\Psi}$, query (3) can be expressed as follows

$$\begin{aligned} WORK'(ID, DNO, PNUM) &\leftarrow \pi_{ID, DNO, PNO}(\overline{EMP} \bowtie_{ID=EID} WORK) \\ RESULT &\leftarrow WORK' \frac{ID}{DNO=DNUM} PROJ \end{aligned}$$

This is obviously an improvement since query (3) was beyond the capabilities of \div ,

Our new division operator is also suitable for solving other kinds of universal queries. Let R be a relation with a single number-valued attribute N . The query for the least number in R would typically be solved by means of aggregate functions. But, in fact, this *is* a universal query and can easily be formulated as such in SQL and relational calculus. In the algebra, however, it seems that \div is not up for the task. Once again, $\frac{T}{\Psi}$ comes to the rescue:

$$(\rho_{(M)}(R) \bowtie_{M=N} R) \frac{M}{M \geq N'} (R \bowtie_{N=N'} \rho_{(N')}(R))$$

4 Summary and Conclusion

Our discussion has revolved around certain universal queries and insufficiencies of the original division operator. Because of its special status as *the* counterpart of universal quantification we have remained faithful to Codd's division-approach to universal queries and found that it is still viable. It seems clear that division based research on universal query optimization, such as (Graefe and Cole, 1995) and (Rantzaou et al., 2003), should be extended to cover generalizations of \div .

We have presented generalized division operators. General division is powerful enough on its own to express any query, and hence any universal query. However, observing that many queries are not universal by nature, we seek a relaxed version which corresponds better to pure universal quantification.

In order to operationalize accurately the concept of universal queries we would need a precise definition to start from. Unfortunately, no such definition can be found and consequently we must settle for approximations based on an intuitive understanding of \forall -quantification in logic. We have approached this by attempting to factor out the original set of query operators from general division, thereby reducing the expressive power to exclude non-universal queries from its capability.

The process results in a redistribution of power over the operator-set $\{\frac{T}{\Psi}, \sigma, \times, \cup\}$. This should not be interpreted as diminishing the significance of other operators. Showing this set to be relationally complete serves only to demonstrate that the transformation from general division has not factored out more than intended.

What remains is the generalized division operator $\frac{T}{\Psi}$ which can handle a number of universal queries that are beyond the powers of \div and therefore is a better approximation to a pure \forall -mechanism.

A final remark: Since general division is relationally complete, it would single-handedly provide for an entire query language. But even if the minimum requirement with regard to expressive power is met, we can hardly claim that reasonable expectations concerning user-friendliness are satisfied. Just take a look at the transformation rules in the proof for theorem 3.1: even a simple difference query would turn into an awfully complicated expression. We doubt that a new query language based directly on general division could challenge the dominance of SQL.

References

- Carlis, J. V., 1986. HAS, a Relational Algebra Operator or Divide is not Enough to Conquer, Proceedings of the second international conference on data engineering, IEEE Computer Society
- Codd, E. F., 1972. Relational completeness of data base sublanguages. In: Justin, R. J. (ed) Data base systems, Courant computer science symposia series 6, Englewood Cliffs, N.J., Prentice-Hall
- Dadashzadeh, M., 1989. An improved division operator for relational algebra, Information systems 14(5),
- Date, C. J., 2004. An introduction to database systems, Addison Wesley
- Date, C. J., Darwen, H., 1992a. Into the great divide. In: Date, C. J., Darwen, H., Relational database writings 1989-1991, Addison Wesley
- Date, C. J., Darwen, H., 1994. Divide and Conquer? In: Date, C. J., Darwen, H., Relational database writings 1991-1994, Addison Wesley
- Elmasri, R., Navathe, S. B., 2007. Fundamentals of database systems, Addison Wesley
- Graefe, G., Cole, R. L., 1995. Fast algorithms for universal quantification in large databases, ACM Transactions on database systems, Vol. 20, No 2.
- Kifer, M., Bernstein, A., Lewis, P., 2004. Database systems, Addison Wesley
- Klug, A., 1982. Equivalence of relational algebra and relational calculus query languages having aggregate functions, Journal of the ACM, Vol 29, No. 3
- Libkin, L., 2003. Expressive power of SQL, Theoretical Computer Science 296
- McCann, L. I., 2003. On making relational division comprehensible, ASEE/IEEE Frontiers in education conference
- Rantzaui, R., Shapiro, L. D., Mitschang, B., Wang, Q., 2003. Algorithms and applications for universal quantification in relational databases. Information Systems 28
- Tessem, B., Bjørnstad, S., Chen, W., 2005. Learning collaboration and software development with pair programming, Proceedings of european research workshop on understanding and rethinking the technology-mediated workplace
- Trovåg, A., 2004. Beyond the divide, Master thesis, Department of information science and media studies, University of Bergen
- Ullman, J., 1988. Principles of Database and Knowledge-base systems, vol. 1, Computer Science Press