On the Expressive Power of Query Languages for Relational Databases

Eric C. Cooper

Computer Science Division — EECS
University of California
Berkeley, CA 94720

ABSTRACT

The query languages used in relational database systems are a special class of programming languages. The majority, based on first-order logic, lend themselves to analysis using formal methods. First, we provide a definition of relational query languages and their expressive power. We prove some general results and show that only a proper subset of first-order logic formulas may be used as a practical query language. We characterize this subset in both semantic and syntactic terms. We then analyze the expressive power of several real query languages, including languages based on the relational calculus, languages with set operators and aggregate functions, and procedural query languages.

Since the partial ordering *"is more expressive than"* determines a lattice among relational query languages, the results of the paper may be viewed as determining some of the structure of this lattice. We conclude with some applications of the results to the optimization problem for query processing.

1. Introduction

There have been several studies of the expressive power of relational query languages. Codd [C2] proved the equivalence of relational algebra and relational calculus, and suggested that languages with this degree of expressive power be termed *complete.*

Aho and Ullman [AU] showed the existence of a computable query (the transitive closure of a relation) which relational algebra is incapable of expressing, and proposed an extension of relational algebra with a least fixed point operator.

Chandra and Harel [CH] redefined *complete* to mean capable of expressing all computable queries. They introduced a complete query language QL, which is an extension of relational algebra with iterative and conditional capabilities.

In this paper, we introduce a formal method of comparing the expressive power of query languages. We define a partial ordering by expressive power that makes the set of query languages into a lattice. The results cited above determine two points in this lattice: one point corresponds to languages equivalent to relational algebra, and the other corresponds to complete languages.

The results of this paper establish additional lattice points corresponding to languages based on the relational calculus, languages with set operators and aggregate functions, and procedural query languages.

2. Query languages and expressive power

We adopt the formal definitons of relational database, query, and query language essentially as stated in [CH].

Definition 2.1.

(1) The *universe* is the set of natural numbers, denoted $\mathbf{N}$.

(2) A *relation* of rank m is a finite set $R \subset \mathbf{N}^m$.

(3) Let $\mathbf{n} = \langle n_1, \ldots, n_k \rangle$. A *database* of type $\mathbf{n}$ is a set of relations $\langle R_1, \ldots, R_k \rangle$, such that for each i, R_i is of rank n_i.

(4) The set of all databases of type $\mathbf{n}$ will be denoted $\mathbf{DB_n}$.

(5) A *query* of type $\mathbf{n}$ is a partial function q such that for each $DB \in \mathbf{DB_n}$, $q(DB)$ is either undefined or else a finite relation of finite rank.

(6) A *query language* of type $\mathbf{n}$ is a set L of expressions and a meaning function μ such that for each $e \in L$, $\mu(e)$ is a query of type $\mathbf{n}$.

(7) A *sublanguage* of $\langle L, \mu \rangle$ is a query language $\langle L_0, \mu_0 \rangle$ with $L_0 \subseteq L$ and $\mu_0 = \mu \restriction L_0$.

We can now formalize the notion of expressive power.

Definition 2.2.

(1) The *expressive power* of L is the set $\mu[L] = \{\mu(e) \mid e \in L\}$.

(2) L_1 is *equivalent* to L_2 $(L_1 \simeq L_2)$ iff $\mu_1[L_1] = \mu_2[L_2]$.

(3) L_1 is *less powerful* than L_2 $(L_1 < L_2)$ iff $\mu_1[L_1] \subset \mu_2[L_2]$.

(4) $L_1 \leqslant L_2$ iff $\mu_1[L_1] \subseteq \mu_2[L_2]$.

The next result is simple but useful.

Theorem 2.3. Suppose $L_1 \leqslant L_2$. Then there exists a sublanguage $L_0 \subseteq L_2$ such that $L_0 \simeq L_1$.

Proof: Let $L_0 = \mu_2^{-1} \mu_1[L_1]$. □

3. Complete languages

The next definition is similar to one in [CH].

Definition 3.1. Let $\langle L, \mu \rangle$ be a query language. Then L is:

(1) *bounded above* iff for every $e \in L$, $\mu(e)$ is computable.

(2) *bounded below* iff for every computable query q, there exists an expression $e \in L$ such that $\mu(e) = q$.

© 1982 ACM 0-89791-065-6/82/001/0361 $00.75

(3) *complete* iff it is bounded above and below.

The following theorem follows immediately.

Theorem 3.2. If L_1 and L_2 are complete, then $L_1 \simeq L_2$.

Since it is proved in [CH] that complete languages exist, let Q denote such a query language. The next result is an immediate consequence of the definition.

Theorem 3.3.

(1) L_1 is bounded above iff $L_1 \leqslant$ Q.

(2) L_2 is bounded below iff Q $\leqslant L_2$.

The final result of this section is analogous to the unsolvability of the halting problem.

Theorem 3.4. If $L \geqslant$ Q, then no algorithm exists which can decide for all expressions $e \in L$ whether or not the induced query $\mu(e)$ is a total function.

4. RC-based languages

In this section, we study general properties of query languages based on the relational calculus of [C1].

Definition 4.1. An *RC-based language* is a first-order language L whose predicate symbols include the relations $R_1,\ldots,R_k$.

In practice, L will also include other functions and predicates, such as arithmetic operations and comparisons, whose usual interpretation is clear. Therefore, a database DB determines a unique structure for L, which we also refer to as DB.

Definition 4.2. Let ϕ be a wff of L with free variables $x_1,\ldots,x_m$. For each database DB, define $\mu(\phi)$ to be the set $\{x_1,\ldots,x_m \mid \phi(x_1,\ldots,x_m)\}^{(DB)}$ of elements of $\mathbf{N}^m$ which satisfy ϕ in the structure DB.

$\mu(\phi)$ is thus a function $q(DB)$ defined on $\mathbf{DB_n}$. But q is not necessarily a query, because $q(DB)$ may not be a finite relation for all databases DB. For example, if ϕ is the wff $x = x$, then $\mu(\phi)$ is the constant function $q(DB) = \mathbf{N}$, and $\mathbf{N}$ is not a finite relation.

One solution to this problem is to agree to call $q(DB)$ undefined whenever it is not a finite relation. Since a query need only be a partial function, any RC-based language L may be regarded as a query language L^*.

Theorem 4.3. If L is an RC-based language which includes arithmetic, then $L^* \geqslant$ Q.

Proof: A result due to Gödel states that the first-order language of arithmetic is capable of representing all recursive functions, and hence by Church's thesis, all computable queries. □

The previous theorem and Theorem 3.4 show that for an RC-based language to be used in a real database system, not all wffs of the first-order language may be allowed in the query language. We now define a class of formulas, called *permissible wffs,* whose induced queries may be evaluated in finite time. Because several later theorems will make use of syntactic properties of these permissible wffs, the definition is rather detailed.

Definition 4.4. Let ϕ be a wff in prenex normal form, with matrix in disjunctive normal form $\bigvee_i \bigwedge_j \phi_{ij}$.

(1) A *direct constraint* on x is a formula $R(\ldots,x,\ldots)$, where R is a relation.

(2) An *indirect constraint* on x is a formula $x = t$, where t is a term and all variables occurring in t are directly constrained (see below).

(3) A free or existentially quantified variable x is *constrained* iff in every disjunct $\bigwedge_j \phi_{ij}$ in which x occurs, some ϕ_{ij} is a direct or indirect constraint on x. The *total constraint* on x is the disjunction of these constraints.

(4) A universally quantified variable x is *constrained* iff in some disjunct $\bigwedge_j \phi_{ij}$ in which x occurs, every ϕ_{ij} in which x occurs is the negation of a direct or indirect constraint on x. The *total constraint* on x is the disjunction of these (positive) constraints.

(5) ϕ is *permissible* iff every variable in ϕ is constrained (or else appears free in a set term--see Definition 5.3).

This definition gives a syntactic characterization of the semantic notion of *safe formula* in [U], since the truth or falsehood of a permissible wff ϕ may be determined from the truth values of a finite number of instances of the matrix of ϕ. More specifically, suppose ϕ is

$$(Q_1x_1)\cdots(Q_mx_m)\psi(x_1,\ldots,x_m,x_{m+1},\ldots,x_n)$$

Let D_i be the finite domain which satisfies the total constraint on x_i. Then the truth of ϕ depends only on the truth of ψ in the finite universe $D_1 \times \cdots \times D_n$.

Theorem 4.5. If ϕ is a permissible wff, then the query $\mu(\phi)$ is a total function.

Proof: With the notation as above, we have

$$\{x_{m+1},\ldots,x_n \mid \phi(x_{m+1},\ldots,x_n)\} \subseteq D_{m+1} \times \cdots \times D_n,$$

which is finite. □

The following lemma will be useful later.

Lemma 4.6. Let $(\forall y_1)\cdots(\forall y_k)\phi$ be a permissible wff. Then there exist permissible wffs ϕ_1 and ϕ_2 in which $y_1,\ldots,y_k$ occur free, such that $(\forall y_1)\cdots(\forall y_k)\phi$ is equivalent to $\phi_1 \leftrightarrow \phi_2$.

Proof: We prove the result only when ϕ is quantifier-free; the general case follows easily by induction.

Let P_0 be the conjunction of the total constraints in ϕ on the y_i, and define P_n inductively to be the conjunction of the total constraints in ϕ on the variables which occur in P_{n-1}. Clearly there exists some n such that P_{n-1} is equivalent to P_n. Let ϕ_1 be $P_0 \wedge \cdots \wedge P_{n-1}$, and let ϕ_2 be $\phi_1 \wedge \phi$. Then $\phi_1 \leftrightarrow \phi_2$ is equivalent to $\phi_1 \rightarrow \phi$, which is equivalent to $(\forall y_1)\cdots(\forall y_k)\phi$ by the remarks following Definition 4.4. □

5. Specific RC-based languages

In this section, several RC-based languages will be presented. The definitions will actually specify only the underlying first-order language; in each case, the corresponding query language is formed from the set of permissible wffs.

We first define the language RC, an extended domain relational calculus in the terminology of [U].

Definition 5.1. RC is the first-order language of arithmetic $\langle +, \cdot, =, < \rangle$ together with constant symbols 0,1,2,... and relation symbols $R_1,\ldots,R_k$.

It also convenient at this point to define a family of sublanguages of RC.

Definition 5.2. For $n \geqslant 0$, RC$_n$ consists of all wffs of RC with no more than n blocks of universal quantifiers in their prenex normal forms. (A *block* is a string of adjacent quantifiers of the same type.)

We note that RC$_0$ consists of the existential wffs of RC. Also, for $m < n$ we have RC$_m \subset$ RC$_n$, and thus RC $= \bigcup_{n=0}^{\infty}$ RC$_n$.

At this point, we wish to introduce the language QUEL of [HSW] into our framework. In order to do so, we assume that QUEL consists only of **retrieve** statements from relations over N, so that it conforms to the definition of query language in 2.1. Also, we restrict the arithmetic of QUEL to addition and multiplication, so that it will be comparable to RC. Finally, we give QUEL a **product** aggregate, analogous to **sum**, so that the aggregate functions are consistent with the arithmetic ones.

It is proved in [U] that pure domain relational calculus is equivalent to pure tuple relational calculus. The same proof shows, *mutatis mutandis*, the equivalence of QUEL as defined in [HSW] with the version we now define. We adopt the more set-theoretical notation of [CB].

Definition 5.3.

(1) If S is a set term of rank n (see below), then for each i, $1 \leqslant i \leqslant n$, $\mathbf{count}_i(S)$, $\mathbf{sum}_i(S)$, and $\mathbf{product}_i(S)$ are *aggregates* of QUEL.

(2) If t is an aggregate of QUEL, then t is a term of QUEL. If t is a term of RC, then t is a term of QUEL.

(3) If ϕ is an atomic formula of RC, then ϕ is an atomic formula of QUEL. If S_1 and S_2 are set terms of equal rank, then the set comparison $S_1 = S_2$ is an atomic formula.

(4) If ϕ is an atomic formula of QUEL, then ϕ and $\neg\phi$ are wffs of QUEL. If ϕ is a wff, then $\exists x \phi$ is a wff. If ϕ_1 and ϕ_2 are wffs, then $\phi_1 \vee \phi_2$ and $\phi_1 \wedge \phi_2$ are wffs.

(5) If R is a relation of rank n, then R is a set term of rank n. If ϕ is a wff with free variables $x_1,\ldots,x_n$, then $\{x_1,\ldots,x_n \mid \phi(x_1,\ldots,x_n)\}$ is a set term of rank n.

Stricly speaking, QUEL is not a first-order language, since it allows set terms. We should therefore specify how a set comparison is to be interpreted. This is an obvious extension of the usual definition of interpretation in a structure, which we dispense with.

Note that (3) above allows only existential quantifiers to occur in QUEL wffs. We also define a family of sublanguages of QUEL.

Definition 5.4. For $n \geqslant 0$, QUEL_n consists of all wffs of QUEL with no more than n levels of nested set terms.

Thus,

$$(\exists z)[R_1(x) \vee R_2(y,z)]$$

is a QUEL_0 wff, while

$$R_1(x) \ \wedge \ \{y \mid R_2(x,y) \wedge \{z \mid R_3(x,y,z)\} = \{z \mid R_4(x,z)\}\} = R_5$$

is a QUEL_2 wff.

The remarks preceding Definition 5.3 apply here: we will consider the languages QUEL_n defined above to be equivalent to the corresponding languages in [HSW]. As with RC, we have $\mathrm{QUEL}_m \subset \mathrm{QUEL}_n$ for $m < n$, and $\mathrm{QUEL} = \bigcup_{n=0}^{\infty} \mathrm{QUEL}_n$.

We note in passing that the occurrence of a free variable in a set term corresponds to a **by** clause in [HSW].

6. The lattice determined by expressive power

Our first theorem follows directly from the definitions of the previous section.

Theorem 6.1. $\mathrm{RC}_0 \simeq \mathrm{QUEL}_0$

The next result is more interesting. It is true, but will not be proved until later, that QUEL is more powerful than RC. Therefore, by Theorem 2.3, there exists a sublanguage of QUEL which is exactly as expressive as RC. We now characterize such a language, which we call QUEL^{set}.

Definition 6.2.

(1) QUEL^{set} consists of all wffs of QUEL which do not contain any aggregate functions.

(2) For $n \geqslant 0$, $\mathrm{QUEL}_n^{set} \triangleq \mathrm{QUEL}_n \cap \mathrm{QUEL}^{set}$.

QUEL^{set} does however allow set terms to be compared for equality (whence the name.)

Theorem 6.3.

(1) For $n \geqslant 0$, $\mathrm{RC}_n \simeq \mathrm{QUEL}_n^{set}$.

(2) $\mathrm{RC} \simeq \mathrm{QUEL}^{set}$

Proof: Since (1) implies (2), we prove only the former, by induction on n. For $n = 0$, the result follows from Theorem 6.1. Assume the result true for $n-1$. Let ψ be a wff of RC_n. It may be written

$$(\exists x_1)\cdots(\exists x_j)(\forall y_1)\cdots(\forall y_k)\phi$$

where ϕ is a wff of RC_{n-1}. Let ϕ' be an equivalent $\mathrm{QUEL}_{n-1}^{set}$ wff. We now invoke Lemma 4.7 to obtain $\mathrm{QUEL}_{n-1}^{set}$ wffs ϕ_1' and ϕ_2' such that $(\forall y_1)\cdots(\forall y_k)\phi'$ is equivalent to $\phi_1' \leftrightarrow \phi_2'$. Let ψ' be the QUEL_n^{set} wff

$$(\exists x_1)\cdots(\exists x_j)[\{y_1,\ldots,y_k \mid \phi_1'\} = \{y_1,\ldots,y_k \mid \phi_2'\}]$$

Then ψ' is equivalent to ψ, which establishes $\mathrm{RC}_n \leqslant \mathrm{QUEL}_n^{set}$.

For the other direction, let ψ' be a wff of QUEL_n^{set}. ψ' may be written

$$(\exists x_1)\cdots(\exists x_j)\phi'$$

where ϕ' is quantifier-free but may contain set term comparisons

$$\{y_1,\ldots,y_k \mid \phi_1'\} = \{y_1,\ldots,y_k \mid \phi_2'\}$$

where ϕ_1' and ϕ_2' are wffs of $\mathrm{QUEL}_{n-1}^{set}$. Apply the inductive hypothesis to obtain equivalent RC_{n-1} wffs ϕ_1 and ϕ_2; the above set comparison is then equivalent to $\phi_1 \leftrightarrow \phi_2$. Let ϕ be the result of substituting $\phi_1 \leftrightarrow \phi_2$ for the original set comparison in ϕ'. Then the RC_n wff

$$(\exists x_1)\cdots(\exists x_j)(\forall y_1)\cdots(\forall y_k)\phi$$

is equivalent to ψ', and the theorem is proved. □

Let us introduce two more QUEL sublanguages, which we call QUEL^{count} and $\mathrm{QUEL}^{count,sum}$.

Definition 6.4.

(1) $\mathrm{QUEL}^{count,sum}$ consists of all wffs of QUEL which do not contain any **product** aggregates or set comparisons.

(2) For $n \geqslant 0$, $\mathrm{QUEL}_n^{count,sum} \triangleq \mathrm{QUEL}_n \cap \mathrm{QUEL}^{count,sum}$.

(3) QUEL^{count} consists of all wffs of $\mathrm{QUEL}^{count,sum}$ which do not contain any **sum** aggregates.

(4) For $n \geqslant 0$, $\mathrm{QUEL}_n^{count} \triangleq \mathrm{QUEL}_n \cap \mathrm{QUEL}^{count}$.

Thus, QUEL^{count} allows only the **count** aggregate, and $\mathrm{QUEL}^{count,sum}$ allows only the **count** and **sum** aggregates.

We may use **count** to simulate universal quantifiers, but not vice versa, as we now show.

Theorem 6.5.

(1) For $n > 0$, $\mathrm{RC}_n < \mathrm{QUEL}_n^{count}$.

(2) $\mathrm{RC} < \mathrm{QUEL}^{count}$

Proof: First we show that for all $n \geqslant 0$, $\mathrm{RC}_n \leqslant \mathrm{QUEL}_n^{count}$. This is very similar to the first part of the proof of Theorem 6.3. The only difference is that we construct a QUEL_n^{count} wff of the form

$$(\exists x_1)\cdots(\exists x_j)[\mathbf{count}(\{y_1,\ldots,y_k \mid \phi_1'\}) = \mathbf{count}(\{y_1,\ldots,y_k \mid \phi_2'\})]$$

To show strict inequality, it suffices to let the database consist of a single relation R of rank 1, and then to show that there is no RC wff $\phi(x)$ equivalent to the QUEL_1^{count} wff $x = \mathbf{count}(R)$. Suppose there exists such a $\phi(x)$ in order to derive a contradiction. Then the total constraint on x is of the form $x = t(y_1,\ldots,y_n) \vee R(x)$, and the total constraint on each y_i is just $R(y_i)$. We may consider t as a polynomial over $\mathbf{N}$ in the variables $y_1,\ldots,y_n$. Since $\phi(x)$ is equivalent to $x = \mathbf{count}(R)$, it must be the case that either $\mathbf{count}(R) \in R$ or else $\mathbf{count}(R) = t(y_1,\ldots,y_n)$ for some $y_1,\ldots,y_n \in R$. Our strategy will be to choose an R such that $\mathbf{count}(R) \notin R$, and infer various properties of the polynomial t. We will then vary R until we obtain contradictory properties of t.

First, let $R = \{0\}$. Since $\mathbf{count}(R) = 1$, we must have $t(0,\ldots,0) = 1$, which shows that t must have a constant term equal to 1.

Now, let $R = \{2\}$. Again $\mathbf{count}(R) = 1$, so we must have $t(2,\ldots,2) = 1$. But this shows that t is identically equal to 1. This contradiction proves the theorem. □

The language QUEL^{count} is less powerful than QUEL^{count,sum}

Theorem 6.6.

(1) For $n > 0$, QUEL$_n^{count}$ < QUEL$_n^{count,sum}$

(2) QUEL^{count} < QUEL^{count,sum}

Proof: Since QUEL$_n^{count}$ is a sublanguage of QUEL$_n^{count,sum}$, we have QUEL$_n^{count} \leqslant$ QUEL$_n^{count,sum}$

To show strict inequality, we proceed as in the proof of the previous theorem. Let the database consist of just R, as before. In order to derive a contradiction, suppose $\phi(x)$ is a QUEL^{count} wff that is equivalent to the QUEL$_1^{count,sum}$ wff $x = \mathbf{sum}(R)$. The total constraint on x is again $x = t(y_1,...,y_n) \vee R(x)$, but here t may involve $\mathbf{count}(R)$. We therefore consider t as a polynomial over N in the variables $y_1,...,y_n$ and $\mathbf{count}(R)$.

First, let $R = \{m+1, m+2, ..., 2m\}$. We have

$$\mathbf{sum}(R) = m^2 + \frac{m(m+1)}{2}$$

and so

$$m^2 < \mathbf{sum}(R) < 2m^2$$

Now $\mathbf{count}(R) = m$, and for each y_i we have $m < y_i < 2m$, so by varying m we can conclude that:

(1) t is of degree 2,

(2) t has only one term of degree 2, and its coefficient is 1.

Now, let p be a prime, and let $R = \{p, 2p, ..., p^2\}$. We see that p divides $\mathbf{sum}(R)$, and p divides all the variables occurring in t (including $\mathbf{count}(R)$). We conclude that p divides the constant term of t. But this is true for all primes p, so the constant term must be 0.

Next, let $R = \{1,2\}$. Since $\mathbf{sum}(R) = 3$, t is forced to have either one or two linear terms.

Finally, let $R = \{2,3\}$. We see that t is always greater than 5, which is a contradiction. Therefore no such $\phi(x)$ exists. □

The language QUEL^{count,sum} is in turn less powerful than QUEL.

Theorem 6.7.

(1) For $n > 0$, QUEL$_n^{count,sum}$ < QUEL$_n$.

(2) QUEL^{count,sum} < QUEL

Proof: Since QUEL$_n^{count,sum}$ is a sublanguage of QUEL$_n$, we have QUEL$_n^{count,sum} \leqslant$ QUEL$_n$.

To show strict inequality, we proceed as before. Suppose $\phi(x)$ is a QUEL^{count,sum} wff that is equivalent to the QUEL$_1$ wff $x = \mathbf{product}(R)$. The total constraint on x is $x = t(y_1,...,y_n) \vee R(x)$, where t is a polynomial over N in the variables $y_1,...,y_n$, $\mathbf{count}(R)$, and $\mathbf{sum}(R)$.

Let $R = \{m+1,...,2m\}$, so that $\mathbf{product}(R) > m^m$. Now $\mathbf{count}(R) = m$, $\mathbf{sum}(R) < 2m^2$, and for each y_i we have $y_i \leqslant 2m$. It follows that for sufficiently large m, $t \leqslant (2m^2)^{k+1}$, where k is the degree of t. But m may be chosen large enough so that $(2m^2)^{k+1} < m^m$, which yields the desired contradiction. □

The final theorem of this section shows that QUEL is not complete.

Theorem 6.8. QUEL < Q

Proof: If QUEL $\geqslant$ Q, then by Theorem 3.4 the problem of deciding whether a QUEL query is a total function would be unsolvable. But this contradicts Theorem 4.5. □

The above proof is non-constructive, because we did not exhibit a particular query which QUEL is incapable of expressing. A constructive proof, analogous to the proof in [AU] of the impossibility of expressing the transitive closure query in relational algebra, would provide a tighter upper bound than just Q on the expressive power of QUEL.

The results of this section may be summarized as follows:

$$\text{RC} \simeq \text{QUEL}^{set} < \text{QUEL}^{count} < \text{QUEL}^{count,sum} < \text{QUEL} < \text{Q}$$

$$\text{RC}_n \simeq \text{QUEL}_n^{set} < \text{QUEL}_n^{count} < \text{QUEL}_n^{count,sum} < \text{QUEL}_n$$

$$\text{RC}_0 \simeq \text{QUEL}_0^{set} \simeq \text{QUEL}_0^{count} \simeq \text{QUEL}_0^{count,sum} \simeq \text{QUEL}_0$$

7. Procedural query languages

In section 6, it was shown that various extensions of RC by aggregate functions and set operations were all strictly less powerful than the complete language Q. It follows from Theorem 2.3 that each of these QUEL-like languages may be translated into a sublanguage of Q. This translation requires a more precise specification of Q than that provided by Theorem 3.2. For instance, by Theorem 4.3 we might take Q to be the set of all (not just permissible) wffs in an RC-based language with arithmetic.

In this section, we will adopt a procedural definition for Q, and we will be interested in the procedural sublanguages corresponding to QUEL-like languages.

Several complete procedural languages have appeared in the literature ([AU], [CB], [CH]). We base our specification of Q on the language introduced in [AU, § 7].

Definition 7.1. The following are programs of Q:

(1) $x := t$, where x is an individual variable and t is a term of RC.

(2) $R := S$, where R is a relation variable and S is a relation variable or constant.

(3) $\mathbf{insert}(\langle t_1,...,t_n\rangle, R)$ and $\mathbf{delete}(\langle t_1,...,t_n\rangle, R)$, where $t_1,...,t_n$ are terms of RC and R is a relation variable of rank n.

(4) **begin** P_1; · · · ;P_n **end**, where $P_1,...,P_n$ are programs of Q.

(5) **if** ϕ **then** P_1 **else** P_2, where ϕ is a quantifier-free wff of RC and P_1 and P_2 are programs of Q.

(6) **for** $\langle x_1,...,x_n\rangle$ **in** R **do** P, where $x_1,...,x_n$ are individual variables, R is a relation variable of rank n, and P is a program of Q.

(7) **while** ϕ **do** P, where ϕ is a quantifier-free wff of RC and P is a program of Q.

Unlike the language of [AU, § 7], Q allows both individual variables and relation variables to change during a **for** loop. We must therefore specify the semantics of (6) carefully. For instance, we wish the following program to compute $\{\mathbf{sum}(R)\}$ in the relation variable S.

```
begin
        $s := 0$;
        $S := \varnothing$;
        for $\langle x\rangle$ in $R$ do
                $s := s+x$;
        insert($\langle s\rangle$,$S$)
end
```

This means that the program **for** $\langle x\rangle$ **in** R **do** P must execute P successively for each element of R. However, the result may be dependent on the order in which this is done, as is the case with the following program.

```
begin
        $n := 0$;
        $S := \varnothing$;
        for $\langle x\rangle$ in $R$ do
                if $n = 0$ then begin
                        insert($\langle n\rangle$,$S$);
                        $n := 1$
                end
end
```

One solution would be to define the effect of **for** $\langle x \rangle$ **in** R **do** P to be the union of the effects of serial iteration over all possible orderings of R. Another approach is to specify that the meaning of order-dependent programs is undefined. This simpler interpretation is sufficient for our purposes.

The query language Q consists of the programs together with the meaning function determined by the semantics of the language. Since rule (7) gives Q the power of a Turing machine, we have the following result.

Theorem 7.2. Q is complete.

Let us now define a sublanguage of Q.

Definition 7.3. RQ consists of all programs of Q which do not contain the **while** construct of rule (7).

Since all programs of RQ halt, we have the following.

Theorem 7.4. RQ < Q

The next theorem follows immediately from [AU, Theorem 3].

Theorem 7.5. RC $\leqslant$ RQ

The main result of this section shows that in fact, RQ is more than powerful enough to express the aggregate functions of QUEL.

Theorem 7.6. QUEL < RQ

Proof: We first show QUEL $\leqslant$ RQ. By the previous theorem, it suffices to show how to simulate the QUEL aggregate functions **count**, **sum**, and **product** in RQ. We do this for **count**; the method for **sum** and **product** is similar. There are two cases to consider.

Case 1:
The argument of **count** is a set term with no free variables.

Example:
count($\{x \mid \phi(x)\}$)

Solution:
By the previous theorem, we may let P be a program of RQ which computes $\{x \mid \phi(x)\}$ and leaves the result in the relation variable R, say. Then the following program computes the above example in the individual variable n.

```
begin
    $P$;
    $n := 0$;
    for $\langle x \rangle$ in $R$ do $n := n+1$
end
```

Case 2:
The argument of **count** is a set term some of whose variables are free (corresponding to a **by** clause in [HSW].)

Example:
$\{x \mid$ **count**$(\{y \mid \phi(x,y)\}) = 1\}$

Solution:
Let P be a program of RQ which computes $\{x,y \mid \phi(x,y)\}$ and leaves the result in R. The following program computes the above example in the relation variable X.

```
begin
    $P$;
    $X := \varnothing$;
    for $\langle x,y \rangle$ in $R$ do
    begin
        $n := 0$
        for $\langle x',y' \rangle$ in $R$ do
            if $x = x'$ then $n := n+1$;
        if $n = 1$ then insert($\langle x \rangle$,$X$)
    end
end
```

We have indicated the general technique whereby an arbitrary QUEL wff may be translated into an equivalent RQ program, and so established QUEL $\leqslant$ RQ.

Strict inequality follows from Theorem 4.5 and the fact that not every RQ program induces a query which is a total function.

□

8. Conclusions

The comparative study of expressive power as outlined in this paper can be used in the design of new query languages. A result which relates the expressive power of a new language to that of an existing one provides a valuable criterion for judging the new language.

Some of the results of section 6 are also applicable to the problem of query optimization. The proofs of Theorems 6.3, 6.5, 6.6, and 7.6 actually yield algorithms for translating a given expression of one query language into an equivalent expression in another. These might be used by an optimizer to change a query with universal quantifiers or set comparisons, for example, into an equivalent query which involves only the more efficient **count** operation.

One promising direction for further research in this area would be to incorporate results on the computational complexity of evaluating various classes of queries.

Acknowledgment

This paper originated as a final project in Philip Bernstein's course in database systems at Harvard. I gratefully acknowledge his many helpful comments and suggestions. Thanks are also due to Joseph Halpern for discussions concerning the model theoretic aspects of this work. An earlier version of this paper appeared as Technical Report TR-14-80, Aiken Computation Laboratory, Harvard University, August 1980.

References

[AU] Aho, A. V. and J. D. Ullman. "Universality of Data Retrieval Languages." *Proc. 6th ACM Symposium on Principles of Programming Languages.* (January 1979), 110-120.

[CB] Casanova, M. A. and P. A. Bernstein. "A Formal System for Reasoning about Programs Accessing a Relational Database." *ACM Transactions on Programming Languages and Systems* **2:3**. (July 1980), 386-414.

[CH] Chandra, A. K. and D. Harel. "Computable Queries for Relational Data Bases." *Proc. 11th ACM Symposium on the Theory of Computing.* (May 1979), 77-90.

[C1] Codd, E. F. "A Relational Model for Large Shared Data Banks." *CACM* **13:6**. (June 1970), 377-387.

[C2] Codd, E. F. "Relational Completeness of Data Base Sublanguages," in *Data Base Systems,* R. Rustin, ed. Prentice Hall (1972), 65-98.

[HSW] Held, G. D., M. R. Stonebraker, and E. Wong. "INGRES — A Relational Data Base System." *Proc. 1975 National Computer Conference.* (May 1975), 409-416.

[U] Ullman, J. D. *Principles of Database Systems.* Computer Science Press (1980).