OPTIMAL IMPLEMENTATION OF CONJUNCTIVE QUERIES IN RELATIONAL DATA BASES

Ashok K. Chandra and Philip M. Merlin
Computer Sciences Department
IBM Thomas J. Watson Research Center
Yorktown Heights, New York

We define the class of conjunctive queries in relational data bases, and the generalized join operator on relations. The generalized join plays an important part in answering conjunctive queries, and it can be implemented using matrix multiplication. It is shown that while answering conjunctive queries is NP complete (general queries are PSPACE complete), one can find an implementation that is within a constant of optimal. The main lemma used to show this is that each conjunctive query has a unique minimal equivalent query (much like minimal finite automata).

1. Example

Given a relation $R \subset D^2$ (as a set of ordered pairs), $|D|=n$, consider the following question "do there exist elements x_1,x_2,x_3,x_4,x_5,x_6 in D such that

$$R(x_1,x_2) \wedge R(x_3,x_2) \wedge R(x_5,x_2) \wedge R(x_1,x_4)$$
$$\wedge R(x_3,x_4) \wedge R(x_5,x_4) \wedge R(x_1,x_6) \wedge R(x_3,x_6)$$
$$\wedge R(x_5,x_6) \ ?"$$

This problem may be solved in the obvious way in time $0(n^6)$. But it can also be answered in constant time. The answer is "true" iff R is a nonempty relation, for if R(a,b) is true, let $x_1=x_3=x_5=a$ and $x_2=x_4=x_6=b$. This is an (albeit contrived) example of the dramatic improvements in speed possible when answering complex queries in a data base.

2. Conjunctive Queries

The relational model of data, introduced by Codd [C70] has gained wide popularity of late [C71a, C71b, C71c, H71, AAFS72, BCKH73, C74, D74, DHP74], and several data base systems have been implemented based on this approach, including PRTV[T76], SEQUEL [CB74], ZETA/TORUS [MST75], INGRES[HSW75], RISS [MM75], RDMS, Tymshare, Query-by-Example [Z75], etc.

Definition. A relational data base

$$B = (D, R_1, R_2, \ldots, R_s)$$

consists of a nonempty finite set D (the domain) and a finite number of relations $R_1,\ldots,R_s$ on D. We will sometimes superscript a relation to indicate its rank. Thus $R_i^p \subset D^p$.

Definition. A first order formula ϕ has the usual quantifiers $\forall$, $\exists$, logical connectives $\vee$, $\wedge$, $\neg$, variables x_i (ranging over the domain D), constants a_i (taken from a possible infinite set of constants A) and relational operators R_i (equality is not allowed).

Note: we are using the usual convention of obfuscating names of relations in formulas, with those in a corresponding model (data base). Constants a_i stand for themselves in the model - this is a slight departure from tradition, and implies that distinct constants stand for distinct elements in the model.

Definition. A first order query is of the form

$$(x_1,x_2,\ldots,x_k).\phi(x_1,\ldots,x_k)$$

where ϕ is a first-order formula which has no free variables other than those in $x_1,\ldots,x_k$. The rank of such a query is k. Note: variables cannot be repeated in the vector $(x_1,\ldots,x_k)$ of the query $(x_i,\ldots,x_k).\phi$.

Definition. A first order query Q and a data base B are said to be compatible if (i) each relation R_i in Q is also in the data base B, and (ii) each constant a_i in Q is an element of the domain D of B.

Definition. Given a query $Q=(x_1,\ldots,x_k).\phi(x_1,\ldots,x_k)$ compatible

with a data base B with domain D, the result $Res(Q,B)$ of Q in B is the set $\{(y_1,\ldots,y_k)\in D^k \mid \phi(y_1,\ldots,y_k)$ is true in B$\}$.

Definition. Two queries Q_1, Q_2 are said to be equivalent $Q_1 \equiv Q_2$ if for every data base B compatible with both Q_1, Q_2, $Res(Q_1,B)=Res(Q_2,B)$.

Lemma 1. The relation $\equiv$ above is indeed an equivalence relation.

Reflexivity and symmetry are immediate. For the proof of transitivity see the appendix.

Definition. A boolean query is of the form

$().\phi()$

having no free variables. Its result is either "true" (the set containing the empty vector) or "false" (the empty set).

Definition. We define a conjunctive query to be a first order query of the form

$$(x_1,x_2,\ldots,x_k).\ \exists x_{k+1}x_{k+2}\cdots x_m.A_1\wedge A_2\wedge\ldots\wedge A_r$$

where each A_i is an atomic formula, i.e. it has the form

$$R_j^p(y_1,\ldots,y_p)$$

where each y is either a variable $x_q, q\leq m$, or a constant a_q.

The following are examples of conjunctive queries:

(i) "List all departments that sell pens and pencils". Given relation

Sales (Department, Item)

the query is

$$(x).\ Sales(x,pen) \wedge Sales(x, pencil).$$

(ii) "Give me all second level and higher managers in department K10. Given the following relation for employees:

Emp (Name, Salary, Manager, Dept.)

the query is

$$(x_1).\ \exists x_2x_3x_4x_5x_6x_7x_8.\ Emp(x_2,x_3,x_4,x_5) \wedge Emp(x_4,x_6,x_1,x_7) \wedge Emp(x_1,x_8,x_9,K10).$$

(iii) "Are there machine shops x,y such that x supplies some part to y and conversely". Given relations

Output (Shop name, item)

and Input (Shop name, item)

the query is

$$().\ \exists x,y,z,w.\ Input(x,z)\wedge Input(y,w) \wedge Output(x,w)\wedge Output(y,z).$$

Conjunctive queries include a large number of queries actually asked in practice, and even when a more general first order query is needed, parts of it are usually conjunctive. For example, if $\vee$'s are allowed as well as $\wedge$'s in the formula, the query can be expressed as a union of conjunctive queries (after converting the formula into disjunctive normal form) which can then be answered separately. In fact, the language Query-by-Example [Z75] is based on a core of conjunctive queries, and it is this subpart that is learnt and used most readily by the so-called naive user [TG75].

3. Generalized join

Definition. $Z_n = \{1,2,\ldots,n\}$.

Codd [C71G] defined a relational algebra including among others, the following operations on relations which are redefined below with some modifications.

(i) Cartesian product. Given vectors $v=(x_1,x_2,\ldots,x_p)$ and $w=(y_1,y_2,\ldots,y_q)$, v.w denotes the concatenation of v,w

$$v.w = (x_1,\ldots,x_p,y_1,\ldots,y_q).$$

The Cartesian product of relations P,Q is

$$P\times Q = \{v.w \mid v\in P \wedge w\in Q\}.$$

(ii) Intersection. Given relations P,Q of the same rank, their intersection is

$$P\cap Q = \{v \mid v\in P \wedge v\in Q\}.$$

(iii) Permutation. Given a permutation

f on Z_p, the permutation of a p-ary relation P with respect to f is

$$\text{Perm}_f(P) = \{(x_1,\ldots,x_p) \mid (x_{f(1)},\ldots x_{f(p)}) \in P\}$$

(iv) Projection. The projection of a relation P of rank p, $p \geq 1$, is

$$\text{Proj}(P) = \{(x_1,\ldots,x_{p-1}) \mid \exists x_p.(x_1,\ldots,x_p) \in P\}$$

(v) Join. Given relations P^p, Q^q and $r \leq p,q$, the join of P,Q is

$$J_r(P,Q) = \{(x_1,\ldots,x_{p-r+q}) \mid (x_1,\ldots,x_p) \in P \wedge (x_{p-r+1},\ldots,x_{p-r+q}) \in Q\}$$

In the special case that r=0, join is the same as Cartesian product,

(vi) Restriction. Given r,s, $1 \leq r < s \leq p$, the restriction of a p-ary relation P is

$$\text{Restrict}_{r,s}(P) = \{(x_1,\ldots,x_{s-1},x_{s+1},\ldots,x_p) \mid (x_1,\ldots,x_{s-1},x_r,x_{s+1},\ldots,x_p) \in P\}.$$

(vii) Selection. Given $r \leq p$, constant $a \in A$ (the set of constants) p-ary relation P, the selection of P is

$$\text{Select}_{r,a}(P) = \{(x_1,\ldots,x_{r-1},a,x_{r+1},\ldots,x_p) \in P\}.$$

(viii) Union Given relations P,Q of the same rank, their union is

$$P \cup Q = \{v \mid v \in P \vee v \in Q\}$$

(ix) Difference. Given the relations P,Q of the same rank, the difference

$$P-Q = \{p \mid p \in P \wedge p \notin Q\}$$

(x) Division. Given a p-ary relation P, $p \geq 1$, and domain D (implicit), the division of P is

$$\text{Div } P = \{(x_1,\ldots,x_{p-1}) \mid \forall x_p \in D, (x_1,\ldots,x_{p-1},x_p) \in P\}$$

Definition. A relational expression is built up from the domain D, the given relations R_i and using the relational operators (i) - (x) above. The compatibility of relational expressions and data bases is as for first order queries. The result Res(E,B) of a relational expression E given a compatible data base B is the value of E where R_i's and D have the interpretations as assigned by B. Two expressions (or an expression and a query) are equivalent ($\equiv$) if these results are equal for every compatible data base. As before, $\equiv$ is provably an equivalence relation.

Codd [C71b], using a formalism different from ours (quantifiers in first order queries are over vectors, not elements; there is no domain D; and certain relations such as =, <, $\leq$ are treated as special) showed that first order queries are essentially the same as relational expressions.

Lemma 2. For each relational expression there is an equivalent first order query, and vice versa.

For the proof see the appendix.

Lemma 3. For each relational expression that uses no operation other than (i)-(vii) above, there is an equivalent conjunctive query, and vice versa.

The proof is similar to that of Lemma 2, and is omitted.

We now define a new operation called generalized join which is a canonical operation for conjunctive queries in the sense that expressions using the generalized join (and selection, restriction) are equivalent to the conjunctive queries.

Definition. Given relations P,Q having ranks p,q respectively, integer r, $0 \leq r \leq p+q$, and injective maps $f: Z_p \to Z_{p+q}$, $g: Z_q \to Z_{p+q}$ such that $Z_r \subset f(Z_p) \cup g(Z_q)$, the generalized join of P,Q with respect to r,f,g is defined as

$$J_{r,f,g}(P,Q)=\{(x_1,\ldots,x_r) \mid \exists x_{r+1},\ldots, x_{p+q}.$$
$$(x_{f(1)},x_{f(2)},\ldots,x_{f(p)})\in P$$
$$\wedge\ (x_{g(1)},x_{g(2)},\ldots,x_{g(q)})\in Q\}$$

The subscripts r,f,g will be deleted when understood.

The generalized join of two relations corresponds to taking the join of the two relations overlapping on specified columns, followed by the deletion of specified columns. For example, given P^4, Q^4, r=3, f(1)...f(4)=1,2,3,5, and g(1)...g(4)=2,3,4,6,

$$J(P,Q)=\{(x_1,x_2,y_2) \mid \exists x_3,x_4,y_4.$$
$$P(x_1,x_2,x_3,x_4)\ \wedge\ Q(x_2,y_2,x_3,y_4)\}$$

(see Fig. 1). A good way of computing this is by first projecting out the fourth columns of P,Q, and then joining the results:

$$P' = \{(x_1,x_2,x_3) \mid \exists x_4.P(x_1,x_2,x_3,x_4)\}$$
$$Q' = \{(y_1,y_2,y_3) \mid \exists y_4.Q(y_1,y_2,y_3,y_4)\}$$
$$J(P,Q) = J(P',Q') = \{(x_1,x_2,y_2) \mid \exists x_3.P'(x_1,x_2,x_3)\wedge Q'(x_2,y_2,x_3)\}.$$

It may be noted that for each fixed x_2, J(P,Q) can be obtained as the result of a matrix multiplication:

$$\text{Let } J(P,Q)_{x_2} = \{(x_1,y_2) \mid (x_1,x_2,y_2) \in J(P,Q)\}$$
$$P'_{x_2} = \{(x_1,x_3) \mid P'(x_1,x_2,x_3)\}$$
$$\text{and} \quad Q'_{x_2} = \{(x_3,y_2) \mid Q'(x_2,y_2,x_3)\}$$
$$\text{Then} \quad J(P,Q)_{x_2} = P'_{x_2} \times Q'_{x_2}$$

where × is boolean matrix multiplication (treating P'_{x_2}, Q'_{x_2} as boolean matrices). Techniques as those in [S69, ADKF70, FM71] may be used for this purpose.

Several authors [T74, R75, G75, NS76] have considered the problem of efficiently computing joins. What has perhaps not been explicitly stated is that techniques for boolean matrix multiplication can be used to advantage for computing joins (and generalized joins). Most of the published algorithms of which we are aware are similar to sparse matrix multiplication schemes.

Let M(a,b,c) be the time required for computing the product of boolean matrices of size a×b and b×c.

Lemma 4. Given relations $P\subset D^p$, $Q\subset D^q$, r, f, g as in the definition of generalized joins, let n=|D|,

$$s=|\{i \mid \forall j.g(j)\neq f(i)\leq r\}|$$
$$t=|\{i \mid \exists j.f(i)=g(j)>r\}|$$
$$u=|\{j \mid \forall i.f(i)\neq g(j)\leq r\}|$$
$$v=|\{i \mid \exists j.f(i)=g(j)\leq r\}|$$

then $J_{r,f,g}(P,Q)$ can be computed on a random access machine in time

$$O(n^p+n^q+n^r) + n^v.M(n^s,n^t,n^u).$$

This lemma provides, in fact, an application for the multiplication of even highly nonsquare matrices. It may be noted that in the special case when s=t=u=v=0 above, the running time can be reduced to O(1) provided our data structure for relations is either a set of vectors, or a boolean array along with a bit to indicate whether or not it is nonempty.

It is easy to see that relational operations (i)-(v) above are special cases of the generalized join. In fact, even selection would be a special case if we had allowed the construct {a} (where $a\in A$ is a constant) as an argument to the generalized join; and restriction would be a special case if we relaxed the condition that functions f,g in the definition be injective.

We extend the definition of a relational expression to allow generalized joins as well as operations (i)-(x).

Lemma 5. Every relational expression containing only generalized joins, restriction and selection is equivalent to some conjunctive query; and every conjunctive query is equivalent to some relational expression containing only generalized joins, restriction and selection, and in

which selection, restriction are never applied to a subexpression containing a generalized join.

The proof is omitted. The second part of the lemma asserts that restriction, selection operations can be propagated to the bottom of the "expression-tree".

4. Complexity Issues

A query may require exponential time to answer if the desired output is large. However, even boolean queries are complex.

Theorem 6. For data bases B and compatible boolean queries Q,
{(Q,B)|Result of Q in B is true}
is logspace complete in polynomial space.

In fact, consider the simple data base $B_0=(D,R)$ where $D=\{0,1\}$, $R=\{0\}$. Then

Theorem 6'. For the data base B_0 and compatible boolean queries Q,
{Q|Result of Q in B is true}
is logspace complete, in polynomial space.

Theorems 6,6' follow readily from the completeness of second order propositional calculus.

Theorem 7. For data bases B and compatible conjunctive boolean queries Q,
{(B,Q) | Result of Q is true in B}
is logspace complete in NP.

Proof. The set is clearly in NP. Its completeness follows from, say, the completeness of the clique problem for graphs. □

Thus boolean first-order queries are PSPACE-complete whereas conjunctive queries are NP-complete. Note that the data base itself is part of the input. Usually, however, queries are substantially shorter than the entire data base. It pays, in such cases, to optimize the query as much as possible, even if such optimization takes a long time with respect to the size of the query (but independent of the data base).

5. Query minimization

The main theorem utilized for optimization (Theorem 12) asserts that for every conjunctive query there is a unique (up to renaming of variables) minimal equivalent query, and it is obtained from the original query by "combining variables". This result is similar to the existence of minimal finite automata, which are obtained by "combining states". Unlike the case for deterministic automata, however, minimizing a conjunctive query is NP-hard. In fact, even checking if two given queries are the same except for renaming of variables, turns out to be logspace equivalent to graph isomorphism.

We assume in the sequel, without loss of generality, that in conjunctive queries atomic formulas cannot be repeated.

Definitions. Let $Q=(x_1,\ldots,x_k)\cdot \exists y_1,\ldots,y_p.A_1\wedge\ldots\wedge A_B$ be a conjunctive query. We define $A_Q=\{a_1,\ldots,a_q\}$ to be the set of constants appearing in Q, $X_Q=\{x_1,\ldots,x_k\}$, $Y_Q=\{y_1,\ldots,y_p\}$. We say Q is trivial if $A_Q=X_Q=Y_Q=\{\}$. The natural model M_Q of Q is the following algebraic structure: the domain of M_Q is $D_Q=X_Q\cup Y_Q\cup A_Q$, and for every relation R^r in Q, $r\geq 0$, the corresponding relation in M_Q has value $\{(z_1,\ldots,z_r)\in D_Q^r \mid R(z_1,\ldots,z_r)$ is an atomic formula in Q$\}$. In addition, M_Q has a finite set of relations taken from the disjoint sets $\{S_1,S_2,\ldots\}$, $\{S_a \mid a\in A$, the set of constants$\}$, which are also disjoint from names for relations in data bases. S_i,S_a stand for unary relations, M_Q contains S_i, $1\leq i\leq k$, having value $\{x_i\}$, and also S_a for each $a\in A_Q$, having value $\{a\}$.

Definition. Given a conjunctive query Q as above, a homomorphism $h:M_Q\to M_Q$, and a subset V, $X_Q\cup A_Q\subset V\subset D_Q$, such that for all $z\in V$, $h(z)=z$, and for all z, $h(z)\in V$, then if Q' is a query such that $M_{Q'}=h(M_Q)$, we say that Q' is a folding of Q.

Definition. If conjunctive queries Q_1, Q_2 are the same except for renaming of variables and reordering of the atomic formulas, and quantified variables, we say that Q_1 and Q_2 are isomorphic.

Note that Q_1, Q_2 are isomorphic iff M_{Q_1} and M_{Q_2} are isomorphic.

Theorem 8. Isomorphism of conjunctive queries is logspace equivalent to isomorphism of undirected graphs.

This result, in a slightly different form, has also been shown by Kozen [K76].

Proof. Graph isomorphism is trivially reducible to the isomorphism of boolean conjunctive queries with a single binary relation R such that if R(x,y) is an atomic formula in the query, then so is R(y,x). For the other direction, we first reduce isomorphism of conjunctive queries to that for undirected node-labeled graphs. For query $Q=(x_1,\ldots,x_k).\phi$, the corresponding labeled graph G is constructed as follows. The set of vertices of G includes (i) $x_1,\ldots,x_k$, labeled $1,\ldots,k$ respectively, (ii) each $a_i \in A_Q$ labeled a_i itself, (iii) each $y_i \in Y_Q$ labeled by the symbol y, (iv) for each atomic formula $A_i=R(z_1,\ldots,z_r)$ in ϕ there are vertices $A_{i,j}$, $0 \le j \le r$ of which $A_{i,0}$ is labeled R, and $A_{i,j}$ is labeled j' for $j \ge 1$, with $A_{i,0}$ connected to each $A_{i,j}, j \ge 1$, and $A_{i,j}$ connected to the node z_j. Isomorphism of labeled graphs is easily reduced to isomorphism of unlabeled graphs - the proof is omitted here. □

The next three theorems assert that foldings are Church-Rosser upto isomorphism, that they preserve equivalence of queries, and that they are hard to find in general.

Theorem 9. If queries Q_1, Q_2 are foldings of Q, then there are isomorphic queries Q_1', Q_2' that are foldings of Q_1, Q_2 respectively.

This follows from Theorems 10, 12 (whose proofs do not use this theorem).

Theorem 10. If Q_2 is a folding of Q_1, then $Q_1 \equiv Q_2$.

Proof. If Q_1 is trivial, so is the theorem. Otherwise let h be the homomorphism folding $Q_1=(x_1,\ldots,x_k).\phi_1(x_1,\ldots,x_k)$ into $Q_2=(x_1,\ldots,x_k).\phi_2(x_1,\ldots,x_k)$. Since $\phi_1 \to \phi_2$ (logical implication) is immediate from the construction, it suffices to show that $\phi_2 \to \phi_1$. Let $\phi_1 = \exists y_1,\ldots,y_p.A_1 \wedge \ldots \wedge A_r$, $Q_2 = \exists y_{i_1},\ldots,y_{i_q}.A_1' \wedge \ldots \wedge A_s'$. If for any compatible database, and any $z_1,\ldots,z_k \in D$, $\phi_2(z_1,\ldots,z_k)$ is true, there exists a mapping $f:\{y_{i_1},\ldots,y_{i_q}\} \to D$ for which $A_1' \wedge \ldots \wedge A_s'$ is true. But extending f to $g:\{y_1,\ldots,y_p\} \to D$ such that $g(y)=f(h(y))$, we see, from the definition of folding, that $\phi_1(z_1,\ldots,z_k)$ is also true.

Theorem 11. $\{(Q_1,Q_2) \mid Q_2$ is a folding of $Q_1\}$ is NP complete. This also holds for boolean conjunctive queries.

Proof. It is clearly in NP. To show it is complete, we reduce the graph coloring problem to it. Given a graph G and integer c, we obtain Q_1, Q_2 such that Q_2 is a folding of Q_1 iff G can be c-colored. There is a single binary relation R. Let V be the set of vertices in G, and $E \subset V^2$ be the symmetric relation for the set of edges. Let C be a set of c variables disjoint from V. We will also use the elements of V for variables below:

$$Q_1 = (). \exists V. \exists C. \left(\bigwedge_{(u,v)\in E} R(u,v)\right) \wedge \left(\bigwedge_{\substack{u,v\in C \\ u\neq v}} R(u,v)\right)$$

$$Q_2 = (). \exists C. \bigwedge_{\substack{u,v\in C \\ u\neq v}} R(u,v).$$ □

The main theorem of this section asserts that for every conjunctive query there is a minimal equivalent query, unique up to isomorphism, that can be obtained from the original query by folding. It should be noted that folding a query cannot increase the number of variables or atomic formulas in a query, and if the number of

variables remains unchanged, the new query is isomorphic to the original.

Theorem 12. For every conjunctive query Q there is a folding Q_0 such that every $Q' \equiv Q$ has a folding Q'_0 isomorphic to Q_0.

We first prove a lemma.

Lemma 13. For conjunctive queries Q_1, Q_2, $Q_1 \equiv Q_2$ iff there exist homomorphisms $R_1: M_{Q_1} \to M_{Q_2}$, and $h_2: M_{Q_2} \to M_{Q_1}$.

Proof. If part. Suppose h_1, h_2 exist. Then given a compatible data base B with domain D, let $(z_1,\ldots,z_k) \in Res(Q_1,B)$ be any element in the result of Q_1. Let the matrix of Q_1 (the conjunct of atomic formulas) be satisfied by the mapping $f: D_{Q_1} \to D$ which is the identity function on A_{Q_1}, and maps the i-th variable in X_{Q_1} into z_i. But then, the matrix of Q_2 is also satisfied by the mapping $g: D_{Q_2} \to D$, $g = f \circ h_2$, since h_2 is the identity function on A_{Q_2}, and maps the i-th variable in X_{Q_2} to the i-th variable in X_{Q_1} (and hence g maps it into z_i), and $|X_{Q_1}| = |X_{Q_2}|$. Thus $(z_1,\ldots,z_k) \in Res(Q_2,B)$, and it follows that $Res(Q_1,B) \subset Res(Q_2,B)$. Similarly $Res(Q_2,B) \subset Res(Q_1,B)$, hence $Q_1 \equiv Q_2$.

Only-if part. Suppose $Q_1 \equiv Q_2$. It is trivial that $|X_{Q_1}| = |X_{Q_2}|$ (since the matrix of a conjunctive query cannot be equivalent to "false"), and easy to see that $A_{Q_1} = A_{Q_2}$, and for all relations R, if R appears in Q_1, it also appears in Q_2. Consider the data base B which is the same as M_{Q_1} (without the special relations S_i, S_a). B is compatible with Q_1, Q_2, and $(x_1,\ldots,x_k) \in Res(Q_1,B)$, where Q_1 is of the form $(x_1,\ldots,x_k).\phi_1$. Let Q_2 be of the form $(x_1',\ldots,x_k).\phi_2$. Since $Q_1 \equiv Q_2$, $(x_1,\ldots,x_k) \in Res(Q_2,B)$, i.e. there is a mapping $h_2: D_{Q_2} \to D_{Q_1}$ (D_{Q_1} is also the domain of B) that is an identity over A_{Q_2}, maps x'_i into x_i for all i, and such that if $R(z_1,\ldots,z_r)$ is an atomic formula in Q_2, $(h_2(z_1)),\ldots,h_2(z_r)) \in R$ in the model B. But h_2 is a homomorphism from M_{Q_1} to M_{Q_2}. Similarly, h_1 exists. □

Definition. For any finite model M, if h is a homomorphism $h: M \to M$ such that for every homomorpism $h': h(M) \to h(M)$, $|h'h(M)| = |h(M)|$, we say h(M) is a minimal submodel of M. Clearly, minimal submodels exist for all finite models, any homomorphism $h': h(M) \to h(M)$ as above is an automorphism, and h(M) is isomorphic to the submodel induced from M on the domain of h(M). It follows readily that the submodel induced from M on the domain of h(M) is itself a minimal model. We will refer to this as a standard submodel.

Proof of Theorem 12. Given queries Q,Q', let Q_0, Q'_0 be queries such that $M_{Q_0}, M_{Q'_0}$ are standard submodels of $M_Q, M_{Q'}$, respectively. From the definitions, Q_0, Q'_0 are foldings of Q,Q'. If $Q \equiv Q'$ then $Q_0 \equiv Q'_0$ by Theorem 10, and by lemma 13, there exist homomorphisms $h_1: M_{Q_0} \to M_{Q'_0}$, $h_2: M_{Q'_0} \to M_{Q_0}$, but since $h_1 \circ h_2$, $h_2 \circ h_1$ must be automorphisms, h_1, h_2 must be isomorphisms. Then by the comment after the definition of isomorphism of queries, Q_0, Q'_0 are isomorphic. □

6. Model and Optimization

Several optimization concepts have been considered for data bases. "Low-level" concepts such as choosing access paths [eg. CHS76] and deciding when to create a new index [eg. C75] are the most common. Several authors have also considered speeding up the computation of joins [T74, R75, G75]. Considering the statement of Theorem 6, however, it seems that "high-level" optimizations and heuristics could result in very large speedups for some queries. [SC75] and [NS76] consider changing the order of evaluation of a query represented as an expression using relations and operators (the latter specifically considers conjunctive queries). These optimizations, however, are usually local, and the overall "structure" of the query remains unchanged. In our optimization we first change the query into a minimal

equivalent query, and then choose (globally) an order of computation. The result is an implementation within a constant of optimal in our model, but which we believe would also be good for practical implementations. Computing such an optimal implementation takes time exponential in the size of the query, but is independent of the data base. One also has the option of ameliorating a query (along the suggested lines) for some given period of time rather than optimizing it completely.

6.1 The Model

Our model of computation is a straight-line program with variables taking relations as values. Statements allowed are (X,Y,Z are variables):

(a) $X \leftarrow R$ (R is a given relation)
(b) $X \leftarrow D$ (D is the domain)
(c) $X \leftarrow Perm_f(Y)$ (a permutation)
(d) $X \leftarrow Restrict_{r,s}(Y)$ (a restriction)
(e) $X \leftarrow Select_{r,a}(Y)$ ("a" is a constant, $r \geq 1$)
(f) $X \leftarrow J_{r,f,g}(Y,Z)$ (a generalized join).

The output is the value of a designated variable, say X_0, at the end of the computation. Further we require that there be no "type-checking" errors in the program - thus a variable cannot be used before it is defined, X_0 must be defined in the program, and r,s,f,g, in (c)-(f) above must satisfy the restrictions in the definitions of the corresponding operations. We say a program P is equivalent to a conjunctive query Q ($P \equiv Q$) if its output (result) equals the result of the query for every data base.

The running time for the program is the sum of the running times for the individual statements. For a data base with domain D, $|D|=n$, the running times are considered as follows. For statements (a)-(e) the running time is zero. For the generalized join operation (f), let p,q,r,s,t,u,v be as in Lemma 4. Then the running time is

$$n^p + n^q + n^r + n^{s+t+u+v} .$$

The total running time of a program P on a data base B is denoted Time (P,B).

Definition. Given a conjunctive query Q, a program P is said to be near-optimal for Q if $P \equiv Q$ and there exists a constant c such that for every program $P' \equiv Q$ and every compatible data base B, $Time(P,B) \leq c.Time(P',B)$.

Discussion of the Model

Statements (a)-(e) are considered to take zero time. These can, however, be implemented in time independent of n (the size of the domain), by using an array representation for arrays, and changing the access functions when an operator is applied. One could change the model by specifying instead, that these operations take constant time, or time n for (b), and n^p for (a), (c), (d), (e), where R,Y have rank p. Either way, the results below apply essentially without change. The running time for the generalized join operation is simply the sums of the lengths of the inputs and output, in addition to the time required for the trivial algorithm for boolean matrix multiplication. One could modify the model in the special case when s=t=u=v=0 (see comment following lemma 4), but again the results below apply with little modification.

The model of computation also does not specifically allow the operations of cartesian product, intersection or projection, as these are obtainable from the generalized join operation.

6.2 Optimization

The running time of any program is a polynomial in n (the size of the domain) with natural numbers as coefficients. The problem of finding a near optimal program for a given query is of finding one whose polynomial has minimal degree (the degree of the polynomial "zero" is -1 by definition). It suffices, therefore, to consider only expressions using operations (a)-(f)

(expressions correspond to programs in which every program variable is used exactly once each on the left and on the right hand side of an assignment, except the output variable X_0 which is used only once on the left hand side). Expressions suffice because for every program there is an equivalent expression whose running time is a polynomial of no higher degree.

Definition. For a program (or expression) P, if T(n) is its running time, then T(n) is a polynomial, and we denote its degree by Deg(P).

Definition. An expression using operators (a)-(f) of Sec. 6.1 is said to have property * if it is of the form

$$Perm(E \times D^k)$$

(where we are using × instead of join for clarity) in which E is an expression containing no Perm or D, and in which no subexpression of a selection or restriction contains a join.

Lemma 14. For each program P there is an expression E having property * such that $E \equiv P$ and $Deg(E) \leq Deg(P)$.

Outline of proof. Represent the program P as a directed acyclic graph, expand it out into an expression (i.e. a tree), delete those occurrences of the domain D which are "joined" with any other "column" of a relation, separate out those occurrences of D that are preserved to the output, and propagate restrictions, selections through joins. □

For each expression $E=Perm(E_1 \times D^k)$ having property * we associate a conjunctive query Q (we write $Q \sim E$) such that Q is equivalent to E, and is constructed from E as in the proof of theorem 2 part 1 (there is a one-one correspondence between the occurrences of atomic formulas in Q and relations in E_1). We extend the relation ~ such that if Q_1, Q_2 are isomorphic and $Q_1 \sim E$ then also $Q_2 \sim E$.

Lemma 15. Given conjunctive queries Q_1, Q_2, and expression E_1 such that Q_2 is a folding of Q_1, and $Q_1 \sim E_1$, then there is an expression E_2 such that $Q_2 \sim E_2$ and for every compatible data base B, $Time(Q_2, B) \leq Time(Q_1, B)$.

Proof is by constructing E_2 from E_1 by deleting those occurrences of relations in E_1 that correspond to atomic formulas in Q_1 which do not occur in Q_2.

We now outline the algorithm for constructing a near optimal program for a given conjunctive query.

Algorithm. Given a conjunctive query Q, find the minimal query $Q_1 \equiv Q$; then for all expressions E, $Q_1 \sim E$, choose the one whose running time is a polynomial of minimal degree. The output is a program that implements this expression.

This algorithm runs in time exponential in the length of the input query (but independent of the data base).

Theorem 16. The above algorithm computes a near-optimal program for the given query.

The proof follows from Theorem 12 and Lemmas 14, 15.

The algorithm can be further speeded up by observing that the expression E can be further restricted such that columns of relations are projected out at the earliest possible moment, i.e. a column is projected out in a join unless it subsequently has to be joined with some other column. This reduces the second part of the algorithm to finding the best order of parenthesizing the matrix of the conjunctive query.

Example. Given query

$$(x).\exists\, u,v,w,y,z.\ R(x,v,w) \wedge R(x,z,w) \wedge S(u,w) \wedge S(u,v) \wedge S(y,w) \wedge S(y,v) \wedge S(y,z).$$

The minimal equivalent query is

$$(x).\exists\, u,v,w.\ R(x,v,w) \wedge (S(u,w) \wedge S(u,v))$$

which can be answered in time $3n^3+4n^2+n$ by computing

$$X \leftarrow \{(v,w) \mid \exists u.\, S(u,w) \wedge S(u,v)\}$$

in time $3n^2+n^3$, and then the output

$$X_0 \leftarrow \{x \mid \exists v,w.\ R(x,v,w) \wedge T(v,w)\}$$

in time $n+n^2+2n^3$.

7. Conclusions

In this paper we introduce the notion of conjunctive queries and define the operation of "folding" such a query. It is shown that every conjunctive query has a unique (upto isomorphism) minimal equivalent query which can be determined by folding. Since foldings are Church-Rosser, it does not matter in what order one finds foldings. This allows the possibility of finding smaller queries equivalent to a given one without having to obtain the minimal. The minimal query can, however, be used to determine a program whose running time is within a constant of optimal for every data base (in the model of computation which allows algebraic operations on relations).

Acknowledgement

The authors appreciate the excellent typing assistance of Marcia Bollard.

Appendix

Proof of Lemma 1. To show that equivalence of queries is transitive, let $Q_1 \equiv Q_2$ and $Q_2 \equiv Q_3$. Let B be any data base with domain D, compatible with Q_1, Q_3. Let $\{a_1,\ldots,a_k\}$ and $\{R_1,\ldots,R_s\}$ be the sets of constants and relations in Q_2 but not in B. Add these to the data base B to obtain data base B' as follows. The domain D' of B' is $D \cup \{a_1,\ldots,a_k\}$. Let a be any fixed element of D. Extend the relations in B to B' as follows. If R is an r-ary relation in B, the corresponding relation R' in B' is defined as follows: for any set $S \in D^s$, let

$E(S) = \{(x_1,\ldots,x_s) \mid \exists (y_1,\ldots,y_s) \in S.$ $\forall i \leq s.$ if $y_i = a$ then $x_i \in \{a, a_1,\ldots,a_k\}$ else $x_i = y_i\}$.

Then $R' = E(R)$. Also let $R_1,\ldots,R_s$ be empty in B'. For any first order formula $\phi(x_1,\ldots,x_m)$ with free variables $(x_1,\ldots,x_m)$, let

$B(\phi) = \{(y_1,\ldots,y_m) \in D^m \mid B \models \phi(y_1,\ldots,y_m)\}$,

and $B'(\phi) = \{(y_1,\ldots,y_m) \in D'^m \mid B' \models \phi(y_1,\ldots,y_m)\}$.

We show by induction on formulas, that $B'(\phi) = E(B(\phi))$ for formulas compatible with B. It is clearly true for atomic formulas $R(y_1,\ldots,y_r)$ where each y_i is a constant or a variable (note: our formulas have no equality or functions). Likewise, it is not difficult to check that if this property holds for ϕ_1, ϕ_2, it also holds for $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, $\neg \phi_1$, $\forall x\ \phi_1$, and for $\exists x\ \phi_1$ (note: quantifiers for B range over D, those for B' range over D'). Hence, $Res(Q_1,B') = E(Res(Q_1,B))$, and likewise for Q_3. But B' is compatible with each of Q_1, Q_2, Q_3, and hence by hypothesis, $E(Res(Q_1,B)) = Res(Q_1,B') = Res(Q_2,B') = Res(Q_3,B') = E(Res(Q_3,B))$. But the function E is one-one. Hence $Res(Q_1,B) = Res(Q_3,B)$. This completes the proof. □

It may be noted that if equality were allowed in first order formulas, then ≡

would no longer be transitive in our formalism (where constants in formulas stand for themselves in the interpretation). For consider the boolean queries $Q_1 = (). \forall x.x=a$. $Q_2 = (). \forall x.x=a \wedge b=b$, $Q_3 = ().$ false. Then $Q_1 \equiv Q_2 \equiv Q_3$, but $Q_1 \not\equiv Q_3$ for they differ on the data base with the single element domain $D=\{a\}$.

This merely points to the pitfalls that may be encountered when formalizing data bases in terms of logic. The alternative would be to allow constants to be mapped into arbitrary elements of the domain by the data base (as in conventional logic). However, as it is usual practice in data base work that constants stand for themselves, one may preface queries by a conjunct asserting that all constants appearing in the query are unequal.

Proof of Lemma 2. Part 1. We show by induction on expressions that for each relational expression there is an equivalent first order query. For expressions D, R^r, the queries are $(x).$ True, $(x_1,\ldots,x_r).R(x_1,\ldots,x_r)$ respectively. For the sequel, let $(x_1,\ldots,x_p).\phi_P(x_1,\ldots,x_p)$ and $(y_1,\ldots,y_q).\phi_Q(y_1,\ldots,y_q)$ be queries equivalent to expressions P,Q. We will implicitly allow renaming variables where appropriate.

(i) The query equivalent to $P \times Q$ is

$$(x_1,\ldots,x_p,y_1,\ldots,y_q).\ \phi_P \wedge \phi_Q$$

(ii), (viii), (ix). For $P \cap Q$, $P \cup Q$, $P-Q$, the queries are, respectively

$$(x_1,\ldots,x_p).\ \phi_P(x_1,\ldots,x_p) \wedge \phi_Q(x_1,\ldots,x_p),$$

$$(x_1,\ldots,x_p).\ \phi_P(x_1,\ldots,x_p) \vee \phi_Q(x_1,\ldots,x_p),$$

and

$$(x_1,\ldots,x_p).\ \phi_P(x_1,\ldots,x_p) \wedge \neg \phi_Q(x_1,\ldots,x_p).$$

(iii), (iv) For $\mathrm{Perm}_f(P)$, $\mathrm{Proj}(P)$, the queries are

$$(x_1,\ldots,x_p).\ \phi_P(x_{f(1)},\ldots,x_{f(p)}),$$

and

$$(x_1,\ldots,x_{p-1}).\ \exists x_p.\phi_P(x_1,\ldots,x_p)$$

(v), (vi), (vii), (x). These follow likewise from their definitions.

Part 2. We show by induction on formulas that for every first order query, there is an equivalent relational expression. Let the query Q be $(x_1,\ldots,x_q).\phi$ and let $\{x_1,\ldots,x_q,\ldots,x_s\}$ be the set of free and bound variables used in ϕ (after renaming all bound variables such that they are all distinct, and distinct from $x_1,\ldots,x_q$). For a subformula ϕ' of ϕ, we construct an expression equivalent to $(x_{i_1},x_{i_2},\ldots).\phi'$, where $x_{i_1},x_{i_2},\ldots$ are the variables not bound in ϕ'.[2] The construction is sketched below.

(i) Let $R(y_1,\ldots,y_r)$ be an atomic formula (where each y_i is a variable x_j or a constant) with t (not necessarily distinct) constants and u distinct variables. One can construct an expression by the application of t selections, r-t-u restrictions, and one permutation to $R \times D^{s-u}$, that is equivalent to $(x_1,\ldots,x_s).R(y_1,\ldots,y_r)$.

(ii) Let E_1, E_2 be the expressions for ϕ_1, ϕ_2, one can construct the expressions for $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$, $\neg \phi_1$ by permutations and projections on E_1, E_2 followed by intersection, union, and difference (from D^k) respectively. Likewise the expressions for $\exists x\ \phi_1$, $\forall x \phi_1$ are obtained from E_1 by permutation, and projection, division respectively. □

References

[AAFS72] M. M. Aastrahan, E. B. Altman, P. L. Fehder and M. F. Senko, "Concepts of a data independent accessing model", Proc. 1972 ACM-SIGFIDET Conference, Denver (Nov. 1972).

[ADKF70] V. L. Arlazarov, E. A. Denic, M. A. Kronrad and I. A. Faradzev, "On economical construction of the transitive closure of a directed graph", Dokl. Akad. Nauk SSSR 194 (1970) 487-488. English translation in Soviet Math. Dokl. 11, 5, 1209-1210.

[BCKH73] R. F. Boyce, D. D. Chamberlin, W. F. King and M. M. Hammer, "Specifying queries as relational expressions", Proc. of ACM SIGPLAN/ SIGIR Interface Meeting on Programming Languages and Information Retrieval, Gaithersburg (Nov. 73).

[C70] E. F. Codd, "A relational model of data for large shared data banks", CACM 13, 6 (June 1970) 377-387.

[C71a] E. F. Codd, "Further normalization of the data base relational model", Courant CS Symposia, 6, Data Base Systems, Prentice-Hall (May 1971).

[C71b] E. F. Codd, "Relational completeness of data base sublanguages", Courant CS Symposia, 6, Data Base Systems, Prentice-Hall, (May 1971).

[C71c] E. F. Codd, "A data base sublanguage founded on the relational calculus", PROC. 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control, San Diego (Nov. 1971).

[C74] E. F. Codd, "Recent investigations in relational data base systems", IFIP74, North Holland, 1974, 1017-1021.

[C75] E. F. Codd, Ed., "Implementation of relational data base management systems", Panel Discussion, NCC (AFIPS) 75, Anaheim (May 1975).

[CB74] D. D. Chamberlin and R. F. Boyce, "SEQUEL" A structured English query language", IBM Technical Report RJ1394 (May 1974).

[CHS76] L. Clough, W. D. Haseman and Y. H. So, "Designing optimal data structures", AFIPS 76, 45 829-837.

[D74] C. J. Date and E. F. Codd, "The relational and network approaches: comparison of the application programming interfaces", Proc. 1974 ACM-SIGFIDET Workshop on Data Description Access, and Control, Ann Arbor, (May 1974).

[DHP74] C. Deheneffe, H. Hennebert, W. Paulus, "Relational model for a data base", IFIP 74, North Holland, 1974, 1022-1025.

[FM71] M. J. Fischer and A. R. Meyer, "Boolean matrix multiplication and transitive closure," 12th SWAT, IEEE (1971), 129-131.

[G75] L. R. Gotlieb, "Computing joins of relations", Proc. ACM SIGMOD Conf. (May 75) 55-63.

[H71] I. J. Heath, "Unacceptable file operations in relational data base", Proc. 1971 ACM-SIGFIDET Workshop on Data Description, Access and Control, San Diego (Nov. 1971).

[HSW75] A. D. Held, M. R. Stonebraker and E. Wong, "INGRES - A relational data base system", AFIPS 75, 44 409-416.

[K76] D. Kozen, "Complexity of finitely presented algebras", Tech. Rep. Cornell Univ., Comp. Sci., TR76-294.

[MM75] D. McLeod and M. Meldman, "RISS - A generalized minicomputer relational data base management system", AFIPS 1975, 44, 397-402.

[MST 75] J. Mylopoulos, J. S. Schuster, D. Tsichritzis, "A multi-level relational system", AFIPS 1975, 44, 403-408.

[NS76] K. E. Niebuhr and S. E. Smith, "Initial implementation of Query by Example", IBM Technical Report (to appear).

[R75] J. B. Rothnie, Jr., "Evaluating inter-entry retrieval expressions in a relational data base management system", AFIPS 75, 44, 417-423.

[S69] V. Strassen, "Gaussian elimination is not optimal", Numerische Mathematik 13 (1969) 354-356.

[SC75] J. M. Smith and P. Y. Chang, "Optimizing the performance of a relational algebra database interface", CACM 8, 10 (Oct. 1975) 568-579.

[T74] S.J.P. Todd, "Implementation of the join operator in relational data bases", IBM U.K. Scientific Centre, Technical Note 15, Peterlee (1974).

[T76] S.J.P. Todd, "The Peterlee Relational Test Vehicle - a system overview", IBM Systems Journal 15, 4, 1976, 285-308.

[TG75] J. C. Thomas and J. D. Gould, "A psychological study of query by example", AFIPS 75, 44, 439-445.

[Z75] M. M. Zloof, "Query by Example", AFIPS 75, 44, 431-438.

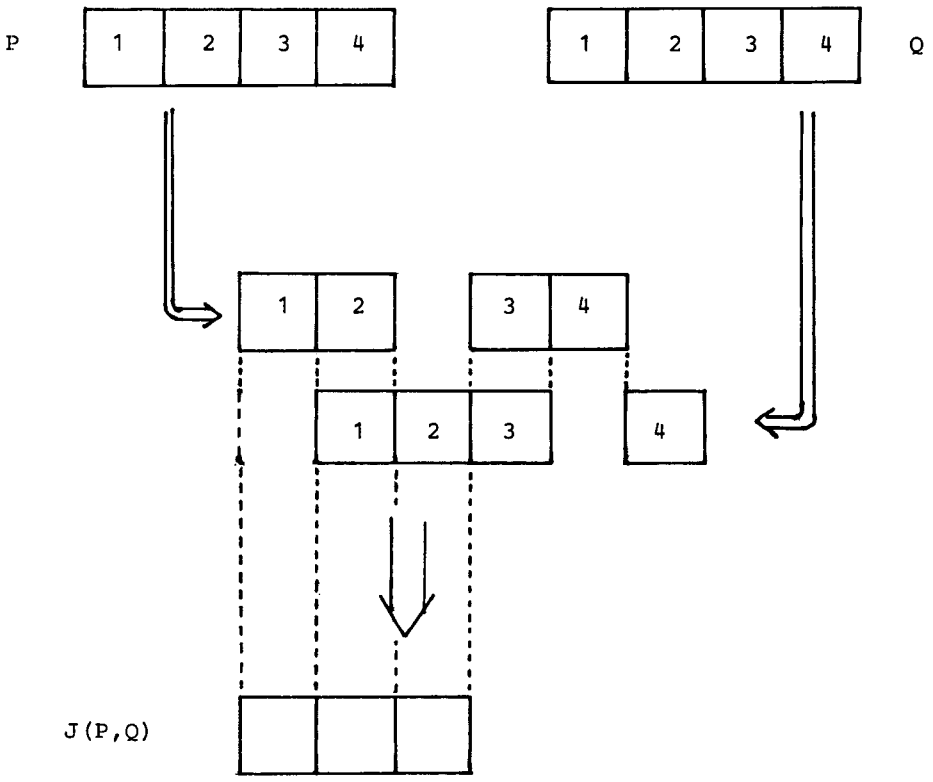

Fig. 1