On query-answering in relational data bases

by E. L. LOZINSKII*

Rutgers University
New Brunswick, New Jersey

INTRODUCTION

In recent years the relational model has been widely adopted for data base description. According to this model a data base, DB, describes certain objects of the world having certain attributes, and the relationships among them. Thus, DB is characterized by a set of attributes, D, a set of domains associated with the attributes, and a set of dependencies, F, corresponding to the relationships among the attributes (all the terms and concepts not defined here are those of References 1-3). A data base is a collection of relations, \( R = \{ R \} \). Each relation \( R_i \) is characterized by a set of attributes \( S_i = \{ D_j | D_j \in D \} \) called its scheme, and consists of a set of tuples. Each tuple is a map from the attributes of the relation scheme to their domains that satisfies all the dependencies of \( F \) (we shall consider functional dependencies).

We shall consider the following model of a data base operation. A data base represented by a collection of relations \( R \) is accessed by a set of jobs submitted by users in order to retrieve, delete, insert or modify any subset of the data. In general, retrieval is an essential part of these processes. Thus, each job accessing a data base requires retrieval of some information from it. Users submit their requests for information in the form of queries, specifying the data which must be retrieved and the conditions that must be satisfied by the desired data. The task of query processing is to determine the set of data to be checked and retrieved from the data base, the proper order in which the data should be accessed and the types of manipulations that must be performed on the data.\(^\text{1} \) This processing is referred to by different authors as query translation\(^\text{2} \) or access path finding.\(^\text{3} \)

Let a query require a retrieval of data which belongs to a set of attributes \( D' \). It can be shown that such a query may be answered in a certain relational data base only if from the same data base a relation can be derived which contains all the attributes of \( D' \) specified by the query. Let us call \( R \)-query a query that requires a creation of relation with a given scheme. \( R \)-queries originate not only from users' requests but can also be initiated by a data base management system (a data base administrator) as a means of relations transformation in the process of data base evolution and restructuring.

Let a query be answerable by means of a procedure which contains a sequence of relational operations project, join, divide and restrict.\(^\text{4} \) Among these operations, the join operation is subject to a number of strong constraints. Specifically, an execution of a join operation on two relations \( R_1 \) and \( R_2 \) with the purpose of creating a new relation \( R_3 = R_1 \times R_2 \) can produce invalid data in the sense that \( R_3 \) can contain a tuple which does not satisfy the original set of dependencies \( F \). In order to avoid the appearance of invalid data, the condition of a lossless join\(^\text{5,6} \) must be satisfied. Thus, we shall concentrate upon join operations of a query-answering procedure, assuming that all the relations produced by joins are properly projected and restricted.

So, considering a collection of relations, \( R \), and a \( R \)-query that requires a relation \( R_k \) with a scheme \( S_k \), the following questions arise:

1. Can \( R_k \) be derived from \( R \)?
2. Which relations should be joined to produce \( R_k \) from \( R \)?
3. What is the sequence of join operations (optimal according to the accepted criteria) that produces \( R_k \)?

The algorithm described in the fourth section answers these questions.

LOSSLESS JOINS

As was shown by J. Rissanen,\(^\text{7} \) a relation \( R_k \) with a scheme \( S_k \) can be produced by means of join and project operations from a set of relations \( R = \{ R_i \} \) (such that \( S_k \subseteq U S_i | R_i \in R \) if these relations satisfy the following conditions for lossless join given in References 9-12.

Let us say that a set of attributes \( B \) functionally depends on a set of attributes \( A \) (abbr. \( A \rightarrow B \)) if there is a functional dependency \( f \in F \) such that \( f: A \rightarrow B \). We shall use the concept of closure, \( C L(X) \), of a set of attributes \( X \) defined in Reference 11 as follows:

1. \( X \subseteq C L(X) \)
2. If \( Y \subseteq C L(X) \) and there is \( f \in F \) such that \( f: Y \rightarrow Z \), then \( Z \subseteq C L(X) \)
3. No attribute is in \( CL(X) \) unless it so follows from (1) and (2). \( X \leadsto Y \) denotes the fact that \( Y \subseteq CL(X) \).

It can be shown that if \( X \leadsto Y \), then the functional dependency \( X \rightarrow Y \) is in \( F \) or can be derived from \( F \) using a set of inference rules based on the axioms given by W. Armstrong. So, the closure of \( X \) is the union of all the sets of attributes dependent on \( X \) according to the adopted inference rules.

A join of relations \( R_i \) and \( R_j \) is lossless (that is, it does not produce any invalid information) iff \( S_i \cap S_j \subseteq S_i \) or \( S_i \cap S_j \subseteq S_j \).

As an example (taken from Reference 11), let us consider a data base characterized by the attributes \( NAME \), \( ADDRESS \), \( PHONE NUMBER \), \( DISTRICT \). We can construct a requested DB with the following information corresponding to the relations \( F \):

\[
\begin{align*}
F &= \{N \rightarrow A, N \rightarrow P, A \rightarrow D, P \rightarrow D\} \\
\end{align*}
\]

By the definition of join operation, two relations, \( R_i \) and \( R_j \), may be joined if they contain common or comparable sets of attributes, \( A \) and \( B \), such that \( A \subseteq S_i \) and \( B \subseteq S_j \). So, four different joins can be executed on the sample relations of \( R \) (common attributes are indicated in parentheses):

\[
\begin{align*}
R_1 &= (\text{R} \times R_2 (N), R_2 = R_3 (A), R_3 = R_4 (D), R_4 = (P D)) \\
R_2 &= (\text{R} \times R_2 (N), R_2 = R_3 (P), R_3 = R_4 (D)) \\
R_3 &= (\text{R} \times R_2 (A), R_2 = R_3 (A), R_3 = R_4 (D)) \\
R_4 &= (\text{R} \times R_2 (P), R_2 = R_3 (P), R_3 = R_4 (D))
\end{align*}
\]

The three former joins satisfy the condition for losslessness, but the latter one does not, because \( CL(\text{ADDRESS}) \neq CL(R) \). The three former joins satisfy the condition for losslessness, but the latter one does not, because \( CL(\text{ADDRESS}) \neq CL(R) \).

Relation \( R_{\text{in}} \), indeed, can contain invalid data, because within a district it relates each phone number to arbitrary addresses not considering that the related phone number and address must belong to the same person name.

Suppose that a relation \( R' \) with a scheme \( S' \) required by a \( R \)-query can be produced losslessly by a sequence of join operations on a set of relations \( \{R_i\} \). Let us denote the operands of a join operation \( R_j (\text{left relation}) \) and \( R_k (\text{right relation}) \), such that if the join is lossless, then without any loss of generality \( S_j \subseteq S_i \subseteq S_k \). A join \( R_j \times R_k \) produces a relation \( R_{\text{out}} \) with a scheme \( S_{\text{out}} = S_j \cup S_k \). There is a subset of attributes \( K_{\text{in}} \subseteq S_i \) (called a key of \( R_i \)), such that

\[
K_{\text{in}} \subseteq S_j
\]

By Armstrong's axioms, i.e. a key of the left relation is also a key of the product. This implies the following property of sequences of lossless join operations:

If a relation \( R' \) with a scheme \( S' \) is produced losslessly by a sequence of join operations from a set of initial relations \( \{R_i\} \) such that \( S' \subseteq S_i \), then there is a relation \( R_{\text{out}} \in \{R_i\} \), called a source relation of \( R' \), such that \( S' \) functionally depends on each key, \( K \), of \( R_j \). (Proofs are omitted here for brevity.)

GRAPH REPRESENTATION

A given set of functional dependencies, \( F \), can be represented by an AND/OR graph, displayed a given set of dependencies \( F=\{f_i\} \) iff for each \( f_j \in F \) such that \( f_j: A \rightarrow B \) (and \( A \) and \( B \) are sets of attributes) all of the following hold:

1. \( V \supseteq \{v_j \mid D_j \subseteq (A \cup B_i)\} \), where \( v_j \) denotes the node displaying \( D_j \).
2. \( v_j \in W \)
3. \( E_i \supseteq \{(v_k, w_i) \mid D_k \subseteq A_i\} \)
4. \( E_k \supseteq \{(w_i, v_m) \mid D_m \subseteq B_i\} \)
5. There are neither other nodes nor other arcs in \( G \).

In words, \( V \) contains nodes, called \( v \)-nodes, corresponding to all the attributes appearing in \( F \) (\( D(v_i) \) denotes the attribute displayed by \( v_i \), \( D(v_i) = D_j \)); \( W \) contains nodes, called \( w \)-nodes, corresponding to all the dependencies appearing in \( F \) (\( f(w_i) \) denotes the dependency displayed by \( w_i \), \( f(w_i) = f_j \)); for each distinct left-side set of attributes \( A_i \) of a dependency \( f_j \) there is a \( w \)-node, \( w_i \in W \), which accepts incoming arcs from all the \( v \)-nodes corresponding to \( A_i \) and emits outgoing arcs to all the \( v \)-nodes corresponding to \( B_i \).

For example, Figure 1 shows a F-graph which displays the following set of dependencies:

\[
\begin{align*}
F &= \{D_1 \rightarrow D_2, D_2 \rightarrow D_3, D_1 \rightarrow D_4, D_1 \rightarrow D_5\}
\end{align*}
\]

Let a collection of relations \( R = \{R_i\} \) represent a DB with a given \( F \). Because our aim is to construct a requested relation from \( \{R_i\} \) we supplement the F-graph \( G = (V, W, E_1, E_2) \) with the following information corresponding to the relations of \( \{R_i\} \): for each \( R_i \) add to \( G \) a node \( u_i \), called a \( w \)-node, such that if \( K \) is one of the keys of \( R_{i} \), then \( u_i \)
emits outgoing arcs to all the nodes of G displaying attributes of K. R_i is displayed by u_i; K is said to be tied by u_i.

We shall say that an AND/OR graph \(H=(U, V, W, E_1, E_2, E_3)\), called an R-graph, displays a given representation \(R\) of a DB with a given \(F\), iff all of the following hold:

1. \(V, W, E_1, E_2\) are those of the F-graph G.
2. \(U=\{u_i | R_i \in R\}\), \(u_i\) denotes the node displaying \(R_i\).
3. \(E_2=\{(u_i, v_k) | D_k \in K_i\}\), \(K_i\) denotes a key of \(R_i\).
4. There are neither other nodes nor other arcs in \(H\).

As an example, Figure 2 shows a R-graph displaying the following representation of a DB having the F-graph of Figure 1:

\[ R_1=(D_1D_2D_3D_4D_5D_6D_7D_8) \]
\[ R_2=(D_2D_3D_4D_5) \]
\[ R_3=(D_1D_3D_4D_7D_9) \]

Let us say that a node \(v_i \in V\) is reachable from a set of nodes \(\bar{V}_V\) (abbr. \(\bar{V}=v_i\)) if the following holds:

a. \(v_i \in \bar{V}\), or
b. There is a node \(w_j\) preceding \(v_i\), \((w_j, v_i) \in E_2\), s.t. all the nodes preceding \(w_j\), \({v_k | (v_k, w_j) \in E_1}\), are reachable from \(V\).

Because each dependency of \(F\) is displayed by the corresponding w-node of \(H\), reachability is equivalent to belonging to a closure in the same sense as \(H\) is equivalent to \(F\). This implies the following:

(i) A node \(v_i \in V\) is reachable from a set of nodes \(\bar{V}_V\) iff \(D(\bar{V}) \supseteq D(v_i)\).
(ii) A node \(v_i \in V\) is reachable from a node \(u_j \in U\) iff \(u_j\) ties a set of nodes \(\bar{V}_V\) such that \(\bar{V}=v_i\).
(iii) If a relation \(R'\) with a scheme \(S'\) can be produced losslessly from a set of relations \(\{R_i\}\), then a R-graph \(H\) which displays \(\{R_i\}\) contains a w-node, \(u_i\), called a source node of \(S'\), from which all nodes \(\{v_k | D_k \in S'\}\) displaying the attributes of \(S'\) are reachable. Indeed, let \(R_m\) be a source relation of \(R'\), then the node \(u_m\) displaying \(R_m\) in \(H\) is a source node of \(S'\).

**JOIN PROCEDURE FOR R-QUERY**

If a relation \(R'\) can be produced from a set of relations \(R=\{R_i\}\) there is, in general, a number of different sequences of join operations producing \(R'\). For example, if a data base having the F-graph shown in Figure 3 is represented by the relations \(R_1=(D_1D_2), R_2=(D_1D_3), R_3=(D_2D_4)\), then a relation \(R'=(D_2D_3D_4)\) can be produced by the following join sequences:

a. \(J_1: R_1'=R_1 \bowtie R_2(D_1), R_1'=R_1 \bowtie R_3(D_1), R_1'=R_1 \bowtie R_4(D_1)\)

b. \(J_1: R_1'=R_1 \bowtie R_2(D_1), R_1'=R_1 \bowtie R_3(D_1), R_1'=R_1 \bowtie R_4(D_1)\)

In these sequences we do not care about project operations which should appear in practical procedures. For instance, in Sequence b the relation \(R_1'\) probably should be projected in \((D_1D_4)\) before \(J_2\). We assume that relations \(R_1'\) produced by joins are properly projected.

One of the join sequences producing \(R'\) should be preferred to others according to certain criteria, one of which, in particular, is the total processing cost. \(CJ=\Sigma c(J_i)\), where \(c(J_i)\) is the processing cost of \(J_i\). The processing cost of a join operation depends on the structure, composition and location of the joined relations and of the product as well. We assume that each relation \(R_i\) is assigned an access cost \(c(R_i)\) (say, proportional to its physical size).

Our goal is to construct an efficient sequence of joins \(J(R')=\{J_1, J_2, \ldots \}\) which produces a relation \(R'\) (with a scheme \(S'\), required by a R-query) from a given set of relations \(R\) representing a data base with a R-graph \(H\). The features of lossless joins of relations and the graph representation described in the second and third sections provide a basis for an algorithm which performs this task in a number of stages as follows.

A. Let \(X\) be a target set, that is a set of nodes displaying the attributes required by a given R-query, such that \(X=\{x | D(x) \in S'\}\). At this stage all the source nodes of \(X\) (from which all the nodes of \(X\) are reachable) are found in \(H\), constituting a set \(U_1\). If \(U_1\) is empty then, by (iii), the requested relation cannot be constructed according to the given data. The algorithm used at this stage is a graph equivalent of the Membership algorithm presented in Reference 15.

---

From the collection of the Computer History Museum (www.computerhistory.org)
B. Let us consider a u-node \( u_i \), a v-node \( v \) and a set of relations \( P(u_i, v) \) such that a relation \( R' \) with a scheme \( S' = D(K_i) \cup D(y) \) can be produced from \( P(u_i, v) \) but cannot be produced from a subset of \( P(u_i, v) \). We shall call \( P(u_i, v) \) a join path from \( u_i \) to \( v \). \( CP(u_i, v) \) stands for the cost of \( P(u_i, v) \) such that \( CP(u_i, v) = \sum (R) | R \in P(u_i, v) \). \( P(u_i, v) \) denotes a join path from \( u_i \) to a set of nodes \( Y \), such that \( P(u_i, Y) = \cup P(u_i, v) | v \in Y \). Stage B finds a join path from \( u_i \in U \) to a given target set \( X \) which has the minimum join path cost among all the join paths from \( u_i \) to \( X \) found by the algorithm. This stage is based on Dijkstra's algorithm for the shortest path in a graph.\(^{16}\)

C. As was mentioned previously, there is, in general, a number of different join procedures (i.e., sequences of join operations), \( J(u_i, X) \) producing \( R' \) from \( P(u_i, X) \). Let \( J^o(u_i, X) \) denote the optimal join procedure that has the minimum total processing cost, \( CJ^o(u_i, X) = \min CJ^o(u_i, X) \) for all the source nodes \( u_i \in U \), an optimal join procedure \( J^o(u_i, X) \) is built. This stage is based on Huffman's algorithm\(^{17}\) for finding a tree with minimum weighted path length. Then among all these join procedures the algorithm chooses the one, \( J(R') \), with the lowest processing cost, such that \( CJ(R') = \min CJ^o(u_i, X) | u_i \in U \). If \( R' \) cannot be produced from the given data, the algorithm returns \( J(R') = \phi \).

### COMPLEXITY ESTIMATION

Let \( \tau \) stand for a measure of complexity (say, the running time) of an algorithm, and \( F \) denote the number of all appearances of attributes in all the dependencies of \( F \) (if all the dependencies belonging to \( F \) are written down as a sequence of names of attributes appearing in them, then \( F \) is the length of this sequence). Let \( |R| \) denote the cardinality of \( R \).

The analysis of the algorithm of the fourth section gives the following estimation of the complexity of its stages:

\[
\tau(A) = O(|R| |F|)
\]
\[
\tau(B) = O(|R|^2 |F|)
\]
\[
\tau(C) = O(|R|^{|F|})
\]

Thus, the complexity of the algorithm finding an efficient join procedure for a given query is polynomial,

\[
\tau = O(|R|^{|F|})
\]

(cf. an exponential-time algorithm given in Reference 12).

### ACKNOWLEDGMENTS

The author is grateful to his anonymous referees for helpful comments on a draft of this paper.

### REFERENCES