ABSTRACT

A new query translation and optimization algorithm is presented. The algorithm is being implemented as the local query translation and optimization technique of Honeywell's Distributed Database Testbed System (DDTS). The algorithm translates local queries expressed in representational schemas (relational) to their equivalent internal schemas (network). The technique is new in that it does not translate each relational command in isolation, but rather attempts to find a collection of relational commands for which an optimized sequence of CODASYL DML commands can be generated. The optimization minimizes the number of disk accesses by taking advantage of the access paths available to the CODASYL local database management systems and the relationship information of the variables used in the relational commands.
INTRODUCTION

In a distributed database system, portions of the data are stored at different nodes in the network. All of the data are regarded as one database, because internode communication and resource sharing permit access to the data resident at other nodes (subject to the access constraints of the network). A system-wide discipline is needed to enforce or facilitate security, data access, resource use, operating procedures, database definition, and data and program transfer among system nodes. At each node, a local database management system provides access to the resident data. These local database management systems may be heterogeneous within the network. It is necessary, therefore, to provide users with a general transaction processing interface that processes user requests while hiding the heterogeneity of the system from the user.

The ANSI/SPARC Study Group on Database Management Systems has proposed a framework consisting of three levels of schema definitions. These levels consist of:

1. **external schema**: the description of the user view
2. **conceptual schema**: the description of the logical view of the data
3. **internal schema**: the local DBMS implementation of the database

In order to model distributed databases, a recent proposal generalizes the ANSI/SPARC framework to five levels (see Figure 1). This proposal extends the internal schema to:

3.1. **global representation schema**: the description of the global representation of data
3.2. **local representation schema**: the description of the database at a given node
3.3. **local internal schema**: the local DBMS implementation of the database.

A different data model may be needed to perform the functions at each different schema level. This means that multischema architectures may entail multimodel architectures. At the external schema level, for instance, different data models may be used to describe different user views of the data. The conceptual schema level requires a semantic data model that is not cluttered by implementation details. The representation schema levels require a data model that possesses powerful data manipulation operators (e.g., the relational data model). The internal schemas are defined in terms of the models supported by the Local DBMSs (e.g., the CODASYL DBTG Model). Thus, in a DDBS that incorporates a multischema/multimodel architecture, various phases of translation are needed in order to express a user view of data in one or more local DBMS representations of that data. Also, in order to achieve reasonable system performance, optimization techniques must be employed in these translations and for access to the data at the local nodes.

The purpose of this paper is to describe a general method of translating and optimizing a database query from the local representation schema level to the local internal schema level. A number of other papers have described methods of query translation and optimization from the external level to the global representational level and from the global representational level to the local representational level. The methods described in these papers and in this one are being implemented in the Distributed Database Testbed System (DDTS) at Honeywell's Corporate Computer Sciences Center. The external level and conceptual level of the database are described by the Entity-Category-Relationship Model. The representational schemas in DDTS use an extended relational model of data. The local database management systems are Honeywell IDS/II systems, which use a CODASYL network model. Thus, the goal of this work is to describe methods of translating and optimizing relational subqueries at local nodes with CODASYL database management systems.

Other work on this problem is taking place at several research centers where distributed database systems are being constructed. The differences in translation and optimization methods among these centers are due to the different data models used at the representational levels and the local internal level of the systems. A recent paper by Dayal and Goodman addresses a translation and optimization environment similar to that of DDTS. A major difference exists in that their proposed methodology interpretively generates a database access strategy for each query entered into the system. A cost formula is optimized in order to derive efficient local processing strategies. Although this method produces an optimal access strategy, the complexity of this derivation is exponential with respect to the query size and thus could be quite costly to execute at run time. In contrast, the object of the methodology in this paper is to rapidly recognize only the query access patterns that are potentially beneficial for optimization. These patterns are preprocessed and are readily available for execution.

The second section discusses the translation of subqueries from the relational data model to the CODASYL network data model. During this translation, the subquery is optimized to take the best advantage of the implemented access paths on the local DBMSs. In the third section we discuss methods for recognizing these optimization opportunities. The fourth section describes the implementation of the translation and opti-
mization methods in DDTS. We conclude the paper by briefly describing planned performance studies of our methodology and suggesting future extensions.

LOCAL TRANSLATION AND OPTIMIZATION

A query, received by a local node for processing (compilation and/or execution), is a list of relational commands on a set of base or temporary relations in a database. Local queries are composed of relational operations that can be executed completely on a local relational schema. Since we assume that the local database management systems are based on CODASYL specifications (IDS/II), the system must transform local relational schemas into local network schemas and relational commands into an equivalent sequence of network DML statements, which can be executed on the network schemas. The transformation of schemas (relational to network and network to relational) is done at database design time and is stored in local data dictionaries for use in later translation and optimization.

This transformation is based on a one-to-one correspondence between relations and record types and between attributes and data items. Stated more formally, the relational to network transformation is:

Let S be a relational schema with K relations, then:

1. For each relation Ri, 1 ≤ i ≤ K define a record type Ni such that (a) Ni contains one data item for each attribute
of Ri, and (b) for every key of Ri, define a key for Ni equal to the key of Ri.
2. For each key of Ri that appears as an attribute of Rj (a foreign key of Rj), define the set Lij between Ni and Nj (Ni owner and Nj member) as optional if the foreign key of Rj could be null, and mandatory if the foreign key of Rj cannot be null.

The transformation of a network schema to a relational schema is done similarly. Note that since CODASYL records do not have to have keys, we may have to use database keys instead. A database key is a system-added item to every record with unique values.

Query translation is done at compile time for compiled queries and at execution time for interactive queries. A compiled query is stored in the data dictionary for execution at a later time. There are two approaches for translating relational commands into network DML statements. The first approach is to translate relational commands one by one, as described in Vassilou and Lochovskyl and Zaniolo. The second approach, used in this paper, is to translate a collection of relational commands as one optimization unit.

In the first translation approach each relational command is mapped into a set of DML commands, which have the same effect as the relational command, but on the network schema. This approach has the drawback of not taking full advantage of the optimization potential of the query being translated. An example can clearly illustrate this point. Consider the following relational schema and query:

SCHEMA:
DEPT(DNAME, HEAD, BUDGET)
STUDENT(SNAME, SSN, DEPT_NAME, SEX)
(DNAME and SNAME are keys, DEPT_NAME is a foreign key)

QUERY:
Print all information about all female students in departments with budgets less than $1,000,000.

The relational solution for this query could be:

T1←SELECT(STUDENT) where SEX = “female.”
T2←SELECT(DEPT) where BUDGET < 1000000.
RESULT←JOIN(T1, T2) where DNAME = DEPT_NAME.

The straightforward translation of these relational commands is a set of DML statements that searches all student records selecting only the female students, searches all department records selecting only the ones that have a budget smaller than $1,000,000, and joins the records from the two resulting sets if they have the same department name. Even though straightforward, this solution does not take into account the fact that there is a one-to-many relationship between relations DEPT and STUDENT. This relationship implies that a department can have many students, but each student belongs to only one department. Using the schema transformation rules previously explained, the relational schema is transformed into the network schema shown below.

DEPT_STUDENT Set

DEPT
STUDENT

Note that in this schema the relationship between the DEPT and STUDENT record types is explicitly shown as a set with DEPT record type being the owner and STUDENT record type as the member (set DEPT_STUDENT). Because of the availability of this set in a CODASYL database, we can combine the translation and optimization of the given query as follows:

1. Locate those department records that have BUDGET < 1000000.
2. Search only the members of these departments, selecting the female students.

Note that this solution does not search the student records that are not members of departments with BUDGET < 1000000. If the number of departments in the database is large, but only a few satisfy the stated condition, then the savings in the search time could be considerable.

Two important points should be noted in the above example. First, there is a specific pattern of relational commands in the query (i.e., SELECT, SELECT, JOIN). Second, there is a set of conditions among the variables (relations) used in the pattern of commands. For this example the conditions are that the DEPT record type must be the owner of a STUDENT record type and the join attribute is the attribute upon which the set type is defined. If either one of these criteria were not satisfied, we could not have optimized the query in this manner. The combination of a pattern and its associated set of conditions on the pattern variables are called a template.

One could find other templates of relational commands, similar to the template in the given example, that can be optimized and translated as a unit. These templates are defined at database design time, based upon the implementation of the local CODASYL databases. Optimized DML code for processing records that match each template is stored as the body of a subroutine, possibly in the data dictionary. The inputs to each subroutine are the relation and temporary relation names used in the commands and the output is a temporary relation that contains the results of each pattern. Having done this, the query translation and optimization procedure must look for patterns in the queries that match a template. Whenever it finds a match, it replaces the pattern with a temporary relation that represents the results of the optimized template. For each template found, a call to the subroutine corresponding to the template is generated in the code that will eventually be executed for the query. We call this code the query strategy table and store it in the data dictionary as well.

The process of looking for more templates is continued until either the query is reduced to a temporary relation or there are no more templates to be found. In the first case, the
The query strategy table contains all of the subroutine calls for execution of the query. In the second case, a one-by-one translation of the unmatched commands left in the query must also be integrated into the query strategy table. We describe how we define templates in our optimization scheme more fully in the next section.

To facilitate the template recognition phase of the optimization at compile time, the relational command list is transformed into a tree, called query tree. A query tree is a binary tree in which the nodes represent base and temporary relations and unary and binary operations on them. Examples of unary operations are SELECT, PROJECT, and UPDATE and examples of binary operations are JOIN, UNION, and INTERSECT. Figure 2 shows the query tree for the example query given above. In this figure DEPT and STUDENT are restricted to produce temporary relations T1 and T2, respectively. A JOIN of the two temporary relations T1 and T2 produces the results in RESULT.

LOCAL OPTIMIZATION

Our optimization approach emphasizes the matching of predefined patterns that can be processed as a unit on a CODASYL database. To illustrate this, we have defined a set of templates shown in Figures 3 to 5. The smallest of these templates consists of three nodes: two unary operations on R1 and R2 that are in a set and one binary operation on the results of the unary operations. In Figure 4, the only binary operations now specified for the templates are JOINs over the attributes that define sets. In this template, R1 is an owner/member of R2, which in turn is an owner/member of R3. Figure 5 shows the largest template defined. Larger and more complex templates can be defined. By increasing the number of templates defined on the local databases, a more thorough optimization can be done. However, the price of adding more templates is the additional storage cost of the patterns and the cost of matching these patterns to the query tree. We believe that a relatively small number of patterns will capture a high percentage of the optimization potential of query trees.

The code that executes a general template of n record types has the form:

```
For each r1 in R1 where P1 (r1)
   Get owner/member from R2
   For each r2 in R2 where P2 (r2)
      Get owner/member from R3
      ... b (r1, r2 ... rn)
      write (results)
   end for.
end for.
```

Some R1's have selection predicates, Pi, whose semantics can help access the Ri in an optimal way if Ri has a CALC key or is indexed. It is worthwhile to exploit the semantics of Pi in order to avoid unnecessary set walking. This is especially important in the case of R1 which is at the topmost level in the program and is, therefore, an entry point into the whole linked structure. The algorithm to determine the best available access path to r1 of R1 is as follows:

1. If the CODASYL schema information gives the access mode of R1 as CALC or INDEX and if the CALC key is a nonexistentially qualified attribute, ai, in P1, then determine whether R1 is accessible using the CALC key or index. If R1 is accessible using the CALC key or index, then store this information in an access strategy structure for R1 for use at execution time.

---

From the collection of the Computer History Museum (www.computerhistory.org)
Local Query Translation and Optimization

2. Else, if the access mode of R1 is the VIA SET mode and if the CALC key or index of an owner of R1 is also a nonexistentially qualified attribute, ai, in P1, then determine, as in step 1, if the owner of R1 is accessible using the CALC key or index. If true, then store this information concerning R1's owner in a strategy table for R1. The r1's will be located at execution time by first hashing or indexing to the owner and then retrieving the r1's.

3. Else, each r1 must be retrieved by sequential access across the realms where R1 resides.

An algorithm for pattern matching is used to match subtrees of the query tree against the predefined templates. In version 0 of DDTS, we have employed a finite state automaton to recognize these optimizable subtrees. The algorithm to generate an optimized query execution strategy is as follows:

1. Use the finite state automaton to match the largest possible pattern contained in the query tree.
2. Generate a call to the appropriate precompiled routine that processes this pattern and store that call in the query strategy table. The input parameters are R1...Rn of the pattern plus the P1's.
3. Replace the recognized pattern with a dummy node (i.e., a SELECT ALL node on resulting temporary relation).
4. Repeat steps 1 to 4 on the reduced tree until no more patterns can be matched.

5. Step through the remaining query tree node by node and translate each relational operation into an operation on the network database. These operations are integrated into the code that calls the predefined routines that process the patterns.

IMPLEMENTATION

We mentioned earlier that the extended relational model of data is used to describe both the global representation schemas and the local representation schemas. Tuples of relations in these schemas are uniquely identified by system-assigned values in surrogate attributes. These surrogate keys also de-
fine the CODASYL sets of the local internal schema in DDTS. During the translation of groups of user queries to their equivalent relational algebra operations, we found that approximately 95 percent of the JOIN operations are over these surrogate keys. This is because the query language at the external schema level in DDTS is a graph-oriented language, GORDAS. Data selection in GORDAS is influenced by the dependencies among entities. An example will illustrate the processing of a transaction or subpart of a transaction in DDTS.

A school library maintains records of overdue books in the following way: Students are grouped according to their departments; each student's overdue collections are recorded against his or her name. This information is modeled by using the extended relational schema as the conceptual model. The local database is, however, a CODASYL network database. Queries are expressed in relational algebra and must be translated and optimized to operate on the database. Here we process a user request to find and list all overdue books (along with the names of the defaulting students) held by graduating seniors (i.e., SEMESTER = 8) in the computer science department.

The conceptual schema is given below. Note that DEPT#, S#, and B# are the surrogate attributes of DEPARTMENT, STUDENT, and OD_BOOKS relations, respectively.

DEPARTMENT (DEPT#, DEPTNAME, DEPTHEAD)
STUDENT (S#, SNAME, DEPT#, SEMESTER)
OD_BOOKS (B#, S#, BNAME)

The transaction in SEQUEL-like form would be:

T1 ← Select DEPARTMENT where DEPTNAME = "Computer Science".
T2 ← Select STUDENT where SEMESTER = 8.
T3 ← Join T1 and T2 over DEPT#.
T4 ← Join OD_BOOKS and T3 over S#.
T5 ← Project T4 over SNAME, BNAME.

The optimized query strategy is generated as follows:

```plaintext
r1 = get_first (DEPARTMENT)
   while (r1 exists) do
      if (r1.DEPTNAME = "Computer Science") then
         r2 = get_first_member (STUDENT)
         while (r2 exists) do
            if (r2.SEMESTER = 8) then
               r3 = get_first_member (OD_BOOKS)
               while (r3 exists) do
                  TEMP = concat(r1, r2, r3)
                  TEMP = retain(BNAME, SNAME).
                  write (TEMP)
                  r3 = get_next_member (OD_BOOKS)
               end while
            end if
         end while
      end if
   end while
r2 = get_next_member (STUDENT)
end while
```

Figure 6 shows the query tree for this user request. Note that the initial selection on OD_BOOKS is a trivial SELECT ALL operation, thus, no temporary needs to be formed. Figure 7 is a Bachman diagram of the CODASYL version of the local internal schema. Up to the final PROJECT operation the query matches the template found in Figure 4. The pattern can be optimized as a unit. Let us assume that the DEPARTMENT record type is calc-ed on DEPTNAME and that the other record types have the VIA SET location mode. In addition we assume the following functions. Function “concat” concatenates records. Function “retain” retains only the named fields and therefore is equivalent to the relational project. Function “write” is self-explanatory. The functions “get_first_member” and “get_next_member” return the first record occurrence and subsequent record occurrences, respectively. The “get_first” function is used to retrieve the DEPARTMENT record(s) and returns them either by sequential scan or by using the CALC or index keys.

PERFORMANCE ANALYSIS

We will analyze the effectiveness of the local optimization techniques described in this paper by monitoring the performance of DDTS. We have designed a detailed study wherein two parameters will be varied. The local optimization can
be turned on and off by a software switch at each DBMS in the system. If no local optimization is performed, then a straightforward relational operation to network operation translation will be performed. The second parameter will be the selection of a query stream to run on the system. By defining different query streams as to their primary processing purpose, we will be able to measure the effect of local optimization on retrieval intensive queries, update intensive queries, insertion/delete queries, and various query mixes. The results of this study will provide valuable information for improving future versions of the local translation and optimization module.

SUMMARY AND FUTURE EXTENSIONS

We have described a general method of translating and optimizing a database query, expressed in relational algebra, to network DML commands. Our optimization approach rapidly recognizes the query access patterns that are potentially beneficial for optimization. This is in contrast to other proposed schemes that translate each individual relational command without regard to the interrelationship of the commands and the variables used in these commands. The algorithm has been implemented as the local query optimization and translation of Honeywell's distributed database testbed system, DDTS.

Our future plans call for

1. Integrating other data models and local DBMSs into DDTS: Our methods of translation and optimization can readily be extended to handle other systems with a network model, systems with a hierarchical data model, and systems based on the relational data model. Pattern matching and predefined access path recognition will remain the critical features of the methodology.

2. Sharing local optimization information on the network: Data access paths may be reorganized and enhanced (e.g., by adding new indexes) at a local DBMS. This information should be broadcast on the network. The nodes that handle the distributed query optimization can then send subqueries to the most effective nodes for local translation and optimization.

3. Designing better methods of handling procedural constraints in queries: Pattern matching is difficult when decision structures are included in a query. In the third section, we proposed a simple, but less than satisfactory, method of dealing with this problem. Any pattern that could not be directly matched was translated operation by operation for processing in the local DBMS. We are studying ways to make pattern matching more general in future versions of DDTS.

4. Using a more powerful pattern recognition algorithm to match patterns: The patterns can be matched using a recursive procedure that will recognize any pattern of any complexity. So, in the future, such a recursive algorithm will replace the simple finite state automaton we have used. This means that the pattern recognition algorithm will also generate code for processing any pattern recognized.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under Grant #ECS-8007683.

REFERENCES


