Stanford Heuristic Programming Project
Memo HPP-79-30

December 1979

Computer Science Department
Report No. STAN-CS-79-781

EXPLORING THE USE OF DOMAIN KNOWLEDGE FOR QUERY PROCESSING EFFICIENCY

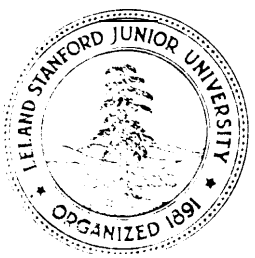
by

Jonathan J. King

Research sponsored by

Advanced Research Projects Agency

COMPUTER SCIENCE DEPARTMENT
Stanford University



Stanford Heuristic Programming Project
Memo HPP-79-30

December 1979

Computer Science Department
Report No. STAN-CS-79-78 1

EXPLORING THE USE OF DOMAIN KNOWLEDGE FOR QUERY PROCESSING EFFICIENCY

by

Jonathan J. King

ABSTRACT

An approach to query optimization is described that draws on two sources of knowledge: real world constraints on the values for the application domain served by the database; and knowledge about the current structure of the database and the cost of available retrieval processes. Real world knowledge is embodied in rules that are much like semantic integrity rules. The approach, called "query rephrasing", is to generate semantic equivalents of user queries that cost less to process than the original queries. The operation of a prototype system based on this approach is discussed in the context of simple queries which restrict a single file. The need for heuristics to limit the generation of equivalent queries is also discussed, and a method using "constraint thresholds" derived from a model of the retrieval process is proposed.

This research was supported by the Advanced Research Projects Agency of the Department of Defense under Contract MDA 903-77-C-0322. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of Stanford University, or any agency of the U. S. Government.

EXPLORING THE USE OF DOMAIN KNOWLEDGE FOR QUERY PROCESSING EFFICIENCY

Jonathan J. King
Computer Science Department
Stanford..University
Stanford, California 94305

Abstract

An approach to query optimization is described that draws on two sources of knowledge: real world constraints on the values for the application domain served by the database; and knowledge about the current structure of the database and the cost of available retrieval processes. Real world knowledge is embodied in rules that are much like semantic integrity rules. The approach, called "query rephrasing", is to generate semantic equivalents of user queries that cost less to process than the original queries. The operation of a prototype system based on this approach is discussed in the context of simple queries which restrict a single file. The need for heuristics to limit the generation of equivalent queries is also discussed, and a method using "constraint thresholds" derived from a model of the retrieval process is proposed.

1. Introduction

An important line of research on databases concerns the use of general statements about a domain, as opposed to the elementary facts usually regarded as the contents of a database. Such general statements (semantic rules; intensional information) can be looked at as conditions that must be true of the facts in the database. Several uses of general statements have been proposed:

-- Integrity constraints. Methods are set up to check that changes to the database involving elementary facts (insertions, deletions, changes in values) do not falsify any of the general statements. ([Mc76], [St75]).

-- Views. General statements are used to define new entities in terms of existing ones in the database. In principle, every database user can have a different "view" of the database consisting of different entities and relationships. ([Ch78], [NG78], [St75]).

-- Deduction. General statements are used during retrieval to provide answers that are not explicitly stored in the database. A single general statement may in effect replace many elementary facts in the database, saving much space. ([MM77], [NG78]).

This paper examines yet another use of general domain statements -- to improve the efficiency of query processing for relational database systems where the query need not specify how the desired data is to be accessed. The general statements are used to produce semantically equivalent descriptions of the set to be retrieved, One of the equivalent descriptions may permit a lower cost of processing than the original.

This paper describes an implemented prototype system that produces the set of equivalent queries permitted by a body of general statements about a domain (also called domain rules). This system, together with a suitable Cost estimator incorporating models of the database structure and processing methods, would yield a low cost equivalent query. The paper also considers how such models might be used to focus the process of selecting transformations on just those general statements that could possibly support transformations to less costly queries.

2. The Query Optimization Probiem -- Standard Approaches

The database query optimization problem is to find a low cost sequence of processing operations needed to retrieve those items meeting the query constraints. in nonprocedural query languages, it is up to the system to perform optimization automatically. (The use of the term "optimization" is inaccurate though historically established; there is no guaranteed procedure for finding the lowest cost processing sequence for arbitrary queries.)

Optimization is needed because retrieval can be unacceptably slow from a very large database, too slow to permit interactive querying in general. Consequently, reasonably involveci planning steps can be tolerated if they result in significant reductions in the actual retrieval cost.

The state of research in query optimization has recently been reviewed in [Ki79]. The standard approach to optimization includes (1) producing a set of expressions that are algebraically equivalent to a given query expression; and (2) for each such expression, determining the lowest cost access paths and processing methods for performing the retrieval. Implemented and proposed optimizers employ various search strategies and heuristics to suggest which rearrangements in the sequence of processing query terms are most likely to produce a low cost sequence of retrieval statements.

This approach to optimization is valuable; virtually every functioning relational database system uses some of these optimization techniques. However, there is a class of situations where these techniques cannot be of sufficient assistance. The following example, based on the experience of a functioning commercial database user, illustrates such a situation.

3. Why Domain Semantics is Needed -- A Motivating Example

A major shipping organization monitors the movements of about 30,000 cargo ships. A maritime insurance association sends it a monthly data tape listing the ports visited by each ship. There are about forty visits per ship per year, or about 1.2 million total visits per year. The organization uses a hierarchical database system. The database contains a file of relatively

permanent ship characteristics such as physical dimensions and cargo capacity. Linked below each ship is a list of its visits; this structure corresponds to the way the data is received and is convenient for processing many common queries. The database also contains a file with data on about 2,000 ports.

The shipping organization supports diverse information needs with this database. Some classes of questions can be answered quickly, such as listing the ports visited by vessels owned by a particular firm, Rapid response for these questions is arranged by setting up structures such as links to speed access to selected items. However, the database structure is poorly suited to process other important questions. For example, to determine the ship name and date of all visits to a port X during a particular year, it is necessary to access each of 30,000 ship records and, for each, to check roughly forty visits to find the visits to port X recorded among them. Such "asymmetrical" access is typical of many databases.

Suppose, however, that there is some characteristic of port X that restricts which ships can visit it. In particular, suppose that port X has a channel depth of only 20 feet, which is quite shallow. A user who knows about X's shallow channel and who knows how the database is organized can transform the query to exploit this knowledge and avoid inspecting every visit record. Rather than ask merely for all visits to port X, the user can ask instead for all visits to X by ships whose draft is less than 20 feet.

Adding the extra constraint as described above replaces a scan of 30,000 ships plus 1.2 million visits with a scan of 30,000 ships plus a number of visits far less than 1.2 million. If only five percent of the ships have drafts of less than 20 feet, then only about 50,000 visits need to be checked. An order-of-magnitude reduction in retrieval effort is obtained by means outside the range of standard optimization techniques.

In our example company, direct usage of the database is confined to a small staff of experts. They routinely use their special knowledge of shipping and of the database structure to perform such "augmentations" of questions from users throughout the company. This kind of knowledge-based optimization can apply to any situation where the person seeking information poses his question in a way that is natural to him without considering how the question can be answered. it may be an essential support for interactive access to very large databases by casual users.

4. The Use of Domain Semantics for Query Optimization

This motivating example suggests the kind of "optimization" that might be developed, one based on knowledge of domain semantics with knowledge of database structure. The domain semantics -- general statements holding for the application domain -- could be embodied in a set of rules expressing relationships among values in the database.. For instance, the domain knowledge used in our example would be that a ship can visit a port only if it is shallower than the port's channel depth.

Such rules are familiar to database researchers. They form a subset of what are commonly called "semantic integrity constraints." The "semantics" they express relate the values of different database attributes to each other. This paper will not be concerned with other kinds of domain semantics discussed in the database literature such as functional dependencies, existence dependencies (insert/delete constraints), and cardinality of relationships among entities.

Conceptually, the strategy of semantic query optimization is the following: use the general semantic rules to infer retrieval set descriptions each of which is "semantically equivalent" to the query description in that it is met by precisely the same set of database items. From the original and the alternative descriptions, select the one which has the lowest estimated processing cost based on a model of the database structure and of the cost of operations given an appropriate cost metric such as the number of page fetches from secondary storage.

The actual operation of a semantic optimization component of a query processing system may have to differ from the conceptual strategy in the interests of overall efficiency. For instance, it may be necessary to generate only those semantically equivalent descriptions that seem most likely to contain a lower cost alternative to the original query. In that event, it would probably be desirable to develop specialized strategies for focussing the attention of the inference mechanism. The following are two examples of such specialized strategies for simple queries:

4.1 Examples of specialized semantic optimization strategies

a. Substitution of indexed for sequential scan

Suppose a query involves retrieval of items from a single relation stored alone on a single file, and the constraints on the query are only on unindexed attributes. If the file has a clustering index on another attribute, try to derive constraints on it and substitute an indexed scan for a sequential scan of the file.

For example, consider a database with information on merchant ships. One of the rules or general statements associated with the database might be that only ships of shiptype "supertanker" are longer than 600 feet. Suppose that a relation with information on SHIPS is stored in a single file which is indexed on the Shiptype attribute, but is not indexed on the Length attribute. Now assume that a user asks for the names of all ships which are longer than 500 feet; that is, for which the Length attribute is greater than 500. Then, using the rule about supertankers, a new constraint in terms of the Shiptype attribute can be added to the original constraint on Length. Instead of a sequential scan, it is now possible to use the index on Shiptype to retrieve all supertankers, and then to check just those ships for their length. If there' are 30,000 ships but only 1,000 supertankers, then only 1/30 as many ship records need be examined for the proper length.

b. Elimination of a join

Suppose a query involves an equijoin of two relations, each stored on its own file (set of data

pages). Both relations have constraints on some attributes, but information is to be listed for the user from only one of them (the "retrieval" file) and not the other (the "constraining" file). Try to derive the 'constraints on the constraining file from those on the retrieval file. If this can be done, the constraints on the constraining file are redundant and can be dropped, and the join can be replaced by a simple restriction on the retrieval file.

For example, suppose there is a database with information about naval ships containing (among other things) the relations SHIPS and OFFICERS, each stored on its own file. One of the rules with this database might be one that states that all carriers are commanded by admirals. Assume the SHIPS relation includes the attributes "Commander" and "Shiptype", and the OFFICERS relation includes the attributes "Officer-Name" and "Rank". Suppose the user asks for the names of all ships with Shiptype "carrier" that are commanded by officers with the Rank of "admiral? This requires an equijoin between SHIPS and OFFICERS on the values of Commander/Officer-Name. However, using the rule mentioned above, it is only necessary to retrieve ail ships with a Shiptype of "carrier? (It would also be advisable to inform the user that a rule had been applied of which he was possibly ignorant.) In this example, a constraint (on Rank) was shown to be unnecessary, as It was derivable from another given constraint (on Shiptype).

5. An Exploratory System for Semantic Query Optimization

A simple exploratory system has been implemented to investigate some aspects of query optimization using domain semantics. It is confined to a much more limited context than the one suggested by the ships/ports/visits example. This section describes the context and operation of this system.

Simply stated, the system starts with a query consisting of a conjunction of constraints on attributes. It systematically derives ail possible additional constraints on those or other attributes using domain rules stated in terms of such constraints. Finally, it determines which of the constraints are necessary to insure that precisely the right items will be retrieved, and which of the constraints are derivable from the others and can therefore be included or excluded from the query depending upon whether or not they make more efficient retrieval possible. The list of equivalent queries is generated from these sets of necessary and optional constraints.

The unguided derivation of all possibie additional constraints followed by the selection of the lowest cost equivalent query is a reasonable strategy only if the original query involves a small number of constraints and there are only a few rules in the system, leading to a manageable number of additional constraints. If, as will often be the case, these conditions are not met, then it is necessary to guide the derivation process toward constraints that are promising with respect to reducing retrieval cost. Experience with the exploratory system has suggested heuristics for guiding the derivation in some simple situations; these will be discussed in section 6.

5.1 Overview -- what the system accomplishes

The exploratory system operates on a small subset of the SODA relational query language [Mo79] developed at SRI International as part of the LADDER system for natural language retrieval from distributed databases [HS78]. In LADDER, the output of the natural language front end is a SODA query in "joinless" format [Sa77]; that is, the database is treated as if it contained just a single relation with all the attributes. This allows the front end to formulate queries without regard for the actual placement of fields on files and therefore without regard for setting up the proper joins between files. This job is left for another LADDER module.

Thus, a joinless query consisting of a conjunction of constraints on different attributes is in fact translatable into a relational query with selections and joins. The exploratory semantic optimizer uses the joinless format. Besides simplifying the queries, the single relation view simplifies the expression of the semantic rules. It also makes the rules independent of structural reorganizations which shift fields to different files. Of course, it is necessary to take the actual placement of fields into account in selecting inexpensive retrieval operations. Even so, derivations can be carried out as if there were a single relation, relying on a component such as the one in LADDER to translate the single relation form of queries into the form reflecting the actual database structure.

The current system can take a simple SODA query in joinless form, can infer additional constraints from the query constraints, and can produce a set of joinless form clueries which are equivalent to the input query in the sense that the very same database items will qualify against any of the queries. The set of queries is formed by separating the constrained attributes into two groups: those whose constraints are entirely derivable from other constraints (called the “optional” attributes) and those for which some part of the constraints comes only from the original query (these are called the “necessary” attributes). One of the queries contains only the constraints on the necessary attributes; the rest of the queries are formed by creating all possible subsets of the optional attributes and adding the corresponding constraints to the first query one subset at a time.

The subject of precisely what queries are semantically equivalent to a given query in a given database is actually more subtle than suggested by the above description. For example, suppose that the query contains the constraint

(LENGTH < 200)

and that it is possible to infer the stronger constraint

(LENGTH < 100).

The current system assumes that the strongest derivable constraint should be the one included among the final set of equivalent queries. Nevertheless, any constraint of the form

(LENGTH < X), X >= 100

could be used instead. The best choice for X would depend on the database structure. For instance, LENGTH might be indexed on ranges of values such as 0 to 150, 150 to 300, and so forth, in which case the constraint

(LENGTH < 150)

might result in a low retrieval cost. Another factor affecting the determination of equivalent queries is the possible existence of more than one derivation for a derived constraint. A thorough examination of semantic equivalence is beyond the scope of this paper.

5.2 Attributes'and constraints

There are two kinds of attributes handled by the system, string and integer numeric. The constraints on a numeric attribute designate intervals on the number line which are possible values for the attribute. For example, a valid numerical constraint would be

(100 < LENGTH < 200).

A string constraint either states that a string attribute is equal to some string constant, or gives a list of string constants which the string attribute does not equal. A valid string constraint would be

(TYPE = "Fishing").

5.3 Queries

The system accepts joinless form SODA queries that contain no disjunctions, quantifiers, or computation operators. In other words, an acceptable query consists of a tuple variable binding expression, selector (output) expressions, and restriction expressions involving constants, all of which refer to a single variable ranging over a single relation. For example, the query "What ships longer than 450 feet are carrying oil?" could be rendered as

((IN S SHIP) ((S CARGO) = "Oil") ((S LENGTH) > 450) (?(S NAME))).

5.4 Rules

Two kinds of rules are used by the system. The first kind is called a "bounding rule". This constrains two attributes with respect to each other. An example of this is a rule stating that "the quantity of cargo carried by a ship cannot exceed the ship's cargo capacity" might be rendered as

(QTY <= CAPAC).

The second kind of rule is called a "production", A production signifies that if some set of constraints (the "antecedents") is satisfied, then another set (the "consequents") is implied. The two sides of a production are called the left hand side (or antecedent) and the right hand

side (or consequent). In the prototype system, the antecedent can be a conjunction of constraints on attributes, and the consequent can be a constraint on another attribute. Thus, a rule stating that "all ships with a capacity of over 150 units (e.g. thousands of barrels) and that are carrying oil will only be sent on voyages of over 4000 miles" could be rendered as

(CARGO = "Oil") & (CAPAC > 150) --> (VL > 4000).

Note that the productions deal only with constants and not with variables. This vastly simplifies and speeds up the matching and application of rules.

5.5 Deriving constraints and formulating equivaient queries -- an example

The derivation process is illustrated with a query on a merchant shipping database: "List the names of tankers that are between 200 and 400 feet tong and are carrying over 150 units of cargo on voyages over 2000 mites tong." Taken as a conjunction of constrained attributes on a single relation and ignoring tuple variables, the query might be viewed schematically as:

(TYPE = "Tanker") & (QTY > 150) & (200 < LENGTH < 400) & (VL > 2000).

The example database is assumed to have the following semantic rules:

Rule 1: "A ship can carry no more cargo than its capacity."

(QTY <= CAPAC)

Rule 2: "Ail tankers with capacity over 100 units are only used in voyages of over 3000 miles."

(TYPE = "Tanker") & (CAPAC > 100) --> (VL > 3000)

Rule 3: "Ail tankers are longer than 350 feet in length."

(TYPE = "Tanker") --> (LENGTH > 350)

The retrieval planning process consists of inferring additional constraints using the rules, determining which constraints are necessary and which are derivable, and generating the set of semantically equivalent descriptions which can be formed into queries.

5.5.1 Make forward inferences

The system starts with the constraints given in the query. It performs all possible forward inferences using domain rules. If there is a finite set of rules and care is taken to insure that no rule is ever executed twice (a production's antecedent may be tested more than once but its consequent can only be asserted once), then this step will terminate.

In the example, the following inferences can be made:

Using rule 1 and the query constraint (QTY > 150), infer (CAPAC > 150).

Using rule 2, the query constraint (TYPE = "Tanker"), and the inferred constraint (CAPAC > 150), infer (VL > 3000)

Using rule 3 and the query constraints (TYPE = "Tanker") and (ZOO < LENGTH < 400), infer (350 < LENGTH < 400).

5.5.2 Identif y derivable constraints

The system now considers every constrained attribute. if any part of the constraints on an attribute comes only from the query constraints given by the user, then it places that constrained attribute in the set of "necessary attributes". If not (meaning that the constraints are entirely derivable from other given or derived constraints), then it places the attribute in the set of "optional attributes".

In the example, the final constraints on TYPE and QTY are taken directly from the query. The constraints on LENGTH have been modified by a rule, but the upper bound is due to the query. The constraint on CAPAC came only from using a rule, and the tightened constraint on VL is also from a rule. Therefore, the set of "necessary" attributes is TYPE, QTY, and LENGTH; while the "optional" attributes are CAPAC and VL. (Recall that this division of attributes into necessary and optional classes is a simplification of the actual task of finding the set of equivalent queries; see section 5.1 above.)

6.6.3 Generate candidate queries

The final set of queries which are candidates for evaluation consists of one query containing just the necessary constrained attributes, plus other queries which add to this every possible subset of the optional attributes. Note that in some cases, the system will not include the original query in the set of candidates. This will occur when some constraint in the original query can be replaced by a stronger constraint on the same attribute.

For our example, the set of candidate queries in schematic form is:

QO -- (TYPE = "Tanker") & (QTY > 150) & (350 < LENGTH < 400).

Q1 -- QO & (CAPAC > 150).

Q2 -- QO & (VL > 3000).

Q3 -- QO & (CAPAC > 150) & (VL > 3000).

S.6.4 Select and carry out the lowest cost candidate query

The final steps are to add the required joins to the joinless candidates, to estimate the cost of carrying out each of the candidate queries, and to evaluate the one with the lowest estimated cost. Although the current system does not perform these steps, components that perform these functions exist in other systems and could presumably be adapted to this system.

6. The Rule Selection Problem

In principle, the process of choosing a low cost query equivalent could be carried out as described above. The method is acceptable if it can be assumed that the problem is usually "small enough": a small number of constraints per query and a small number of rules per attribute. If the assumption is not valid, then the number of constraints and rules will be large enough to lead to the generation of an unacceptably large set of alternative queries. The generation and evaluation steps will both be costly.

To avoid this, an efficient planner should seek to apply only those rules which lead to constraints that can be exploited by retrieval methods that will cost less than the methods that would be used if semantic rephrasing had not been done. The problem can be viewed as a an instance of "heuristic search" through a "problem space" whose "nodes" are sets of constraints on database attributes and whose "operators" are derivation rules embodying domain semantics. The "heuristic evaluation" of the best operator to apply is based on a model of the cost of retrieving data obeying various constraints. A general discussion of heuristic search methods is contained in [Ni71].

We have studied this approach for the simple case of restricting a single relation, where the method of storing the relation is chosen from a small set of alternatives, and there are just a few possible retrieval methods. The storage structures and retrieval methods are patterned on those set forth by Yao and DeJong and are fully described in Appendix A. The set of methods studied includes sequential scan, which requires no auxiliary structures (hence is always applicable), and several methods using an index. Not all the rival methods are always applicable. For example, a clustering indexed scan requires that the file have a clustering index on one of its fields.

6.1 Target attributes

Our method of limiting deductions is to pick the smallest possible set of attributes (or fields, as we assume they are in one-to-one correspondence) as the only worthwhile "targets" for additional constraints. Only rules that can iead to constraints on the target attributes will be used in deductions.

The first heuristic limiting deductions is to include initially in the set of target attributes just those that correspond to an indexed field. That is because, for the limited problem we are studying, the task of the planner is to find a less expensive alternative to sequential scan, and this means a method using an indexed field. (We will ignore the case where a less expensive method than sequential scan is supported in the query constraints; we will assume that this case can be detected and that no deductions will be carried out when it arises.) The target set is further limited to just those attributes for which it appears possible to derive sufficiently strong constraints to make the method they support be less expensive than sequential scan.

We now discuss the necessary supports for the target attribute pruning strategy: a simple

test telling when an indexed method is less expensive than sequential scan; and a check for whether a rule can lead to constraints on a target attribute.

6.2 Cost of rival methods -- the "constraini threshold"

For each method using an index, there is a tradeoff between extra costs incurred retrieving the index, and the reduced costs obtained by going directly to qualifying records in the file. Consequently, sequential scan is the cheapest method when no constraints on indexed attributes can be found, and the cost of each of the rivals to sequential scan directly depends upon how strongly the indexed attribute can be constrained.

The most important observation for our analysis is that for every rival method, there is some value for the strength of the constraint on the indexed attribute at which the rival method becomes less expensive than sequential scan. (Throughout the rest of the paper, the strength of a constraint will sometimes be referred to in terms of its "selectivity." This is a standard term; see [Ya79], for example. The selectivity of a constraint is a number between 0 and 1 giving the fraction of items expected to meet the constraint. Thus, a constraint with selectivity close to 1 is a "weak" constraint, and a constraint with selectivity close to 0 is a "strong" constraint.)

The "constraint threshold" is a simple function of storage parameters. Appendix C lists the expressions giving the constraint thresholds in terms of selectivity for various methods under different storage regimes. The constraint threshold is the basis for a quick test to determine which rival methods could conceivably cost less than sequential scan. An example illustrates the key idea.

6.3 The use of the constraint threshold for selecting rules -- an example

Suppose a query requests all tuples in a relation that meet given constraints on several attributes. Assume the relation is stored as a single file, that none of the fields corresponding to the constrained attributes is indexed, but that the file does have a clustering index on another field F1 and a nonclustering index on yet a different field F2. (See Appendix A for definitions of these index types.)

Because these indices exist, it is possible to retrieve the desired records using clustering indexed retrieval via F1 or nonclustering indexed retrieval via F2. As there are no constraints given on F1 or F2, either method is at this point more expensive than sequential scan. One or both methods will only be less expensive than sequential scan if strong enough constraints can be found on the relevant field or fields. The question is, how strong must such constraints be, and can sufficiently strong constraints possibly be derived from the query constraints?

The answer to the first part of the question is simply that the constraint on F1 must be stronger (have a smaller selectivity value) than the constraint threshold for clustering indexed retrieval, and the constraint on F2 must be stronger than the constraint threshold for nonclustering indexed retrieval. The values of these constraint thresholds (call them T1 and T2 respectively) can be found for the file in question by evaluating simple functions of the file's storage parameters, as given in Appendix C.

Can stong enough constraints be derived on F1 or F2? Suppose that an estimated fraction RSEL of records will pass the given query constraints. The only way to derive new constraints on F1 or F2 is from the existing query constraints. But there is no way to derive a stronger constraint from a weaker one because every item meeting the derived constraints must meet the deriving constraints.

Therefore, if RSEL is greater than T1 then it is impossible to derive a strong enough constraint on Fl, and similarly for T2 and F2. (It should be noted that RSEL is an estimate, so this comparison may give us the wrong answer; still, it appears to be a reasonably good heuristic.)

How can this information be used to prune the derivation process? When the planner gets the query, both F1 and F2 are potential "targets" for additional constraints. Suppose the example file has the "typical" storage parameters given in appendix C. This gives a value of .67 for T1 and a value of .01 for T2. If an estimated one-fourth of the file's records will pass the query constraints, giving an RSEL value of .25, then it may still prove worthwhile to seek constraints on F1, but it will be useless to constrain F2 further (unless of course such constraints are needed to constrain F1 further).

At this stage, then, the only target for further constraints is F1. The planner should only use those rules that can possibly lead to constraints on F1. Any other rules which might be used with the current constraints should nevertheless be left out.

Various methods can be used to determine which rules might lead to constraints on each particular attribute. One straightforward technique, somewhat similar to that employed by the DADM system [KK78], works as follows: Define an "inference path" between attribute A and attribute B as a sequence of rules R1,...,Rm such that given the appropriate constraint on attribute A (and possibly constraints on additional attributes other than B), and applying the R's in sequence, it is possible to derive some constraint on attribute B. Then, for each rule Hi and each attribute Aj, it is possible to determine (for example, when a new rule is added to the system) whether rule Ri is part of some inference path to Aj.

If at some stage of the inference process rule Rx can be applied to the current constraints, it should also be tested to see if any of the attributes to which it is on an inference path is one of the "target" attributes for which further constraints are sought. If the rule fails that test, it should not be used to perform an inference.

In our example, imagine that some production P mentions just a subset of the query constraints in its antecedent. If it lies on an inference path to F1, then the antecedent conditions should be tested and if they are met, the consequent constraints should be asserted. These constraints can be used for further derivations. If P does not lie on an inference path to F1, then no testing should be performed.

7. Summary and Directions for Further Research

A method has been described for using domain semantic rules to find less expensive queries

equivalent to those posed by users who do not take the cost of retrieval into account when posing their queries. The planner using these rules will itself be efficient if queries usually do not' involve a large number of attributes, and if only a small number of rules constrains the typical attribute. A prototype planner that uses all applicable domain rules has been implemented.

If the number of candidate transformations threatens to be large because of the complexity of the query or the size of the rule base, then the generation of transformations must be constrained. An approach to guiding the planner has been identified. It makes use of a model of the structure of the database and the operation of alternative retrieval methods. The central idea is to estimate which attributes could possibly support a less expensive retrieval method if strong enough constraints on them could be derived. The analysis is carried out for the simple case of restricting a single file.

Work is planned along several dimensions to extend the approach introduced in this paper:

1. Completing and testing the current system.

A simple cost evaluator will be used along with existing modules to exercise the system against a functioning database system. The behavior of the planner will be tested as more domain rules are added.

2. Coverage of query types

New strategies are being formulated to handle queries that include joins. This may be extended to other query types that have complex logical structure, such as those that involve computation operators or quantification.

3. Employment of other kinds of domain semantics

The current approach only uses relationships between the values of individual attributes. Semantic optimization strategies based on concept taxonomies and on constraints among the arguments of relationships are being investigated. For instance, a wiclcty applicable class of optimizations would be attempting to "push constraints" higher up a concept hierarchy.

4. Dynamic retrieval planning

The current approach is to perform a single analysis and choose a canciidate query before any part of the retrieval is performed. Methods are being developed to use intermediate retrieval results to reduce further processing costs.

5. Representation and use of knowledge about the completeness of the database

The information in a database often gives only incomplete coverage of the application domain it serves. It is then possible for a user to pose a query about a set of items whose specification

is valid (recognized by the query processor) but about which no data has been stored. It is desirable in such cases to inform the user that the query is well formed but that there is insufficient data to answer it. An approach to this problem is being explored that uses the techniques of rule-based transformations described here.

Appendix A. Storage Organization and Processing Operations

A.1 Basic and auxiliary structures

The basic structure of the database is a data page on which are stored records corresponding to tuples from one or more relations. In this paper, we consider just two kinds of basic storage organization. The simpler case, which will be called the "standard" case, has data pages containing records from only one relation. A more complicated case, the "embedded" case, arises when two relations are related hierarchically in a 1 to N relationship, and the relations are stored to reflect that relationship: one record of the "parent" relation is stored, followed in sequence by the N corresponding records of the "child" relation. Depending upon the page size, the size of the records, and the "fanout" (the number of child records per parent), a single page can contain more or less than a single parent-children grouping. In fact, there can even be pages containing data from only the child relation.

The basic structures described above contain the "actual data" of the database, and are therefore the only structures needed to carry out any retrieval operations. For the sake of efficiency, however, certain "auxiliary structures" can be created.

Here we consider only one kind of auxiliary structure, the "index". An index is a set of pages containing pointers to the records of a file containing particular values for some attribute. An index is usually implemented hierarchically with some number of pointers in each page at any index level. The pointers at higher levels point to index pages at lower levels. The pointers at the leaf nodes of the index tree point to actual data pages. If an index to a relation exists, we can think of the relation as being implemented with two sets of pages: the "basic" pages that contain the actual data, and the "auxiliary" index pages.

A.2 Processing operations

We consider only a small set of processing operations for extracting qualifying records from a particular file. The methods are:

a. Sequential scan (SS): A complete scan of every page in the "file cluster", the set of pages known to contain every record of the file. This method needs no auxiliary structures so it works in every situation.

b. Nonclustering index scan (NCI): An index exists on an attribute but the order of items in the

index does not correspond to the order in the data pages; hence, it is not a "clustering" index. To perform this method, the index is retrieved, the pointers to records with appropriate values are found, and then the pointers are used to fetch the records from appropriate data pages.

c. Clustering index scan (CI): Same as NCI except that the order of items in the index is the same as the order in the data pages.

d. Linked index scan (LI): The constrained relation is the child in a parent-child relationship. It is assumed that (1) the parent relation is stored separately on its own set of pages; (2) there is a clustering index on the child file; and (3) the parent file provides index values for the clustering index in the same sequence as they appear in the index and the child file. In other words, each child record contains an attribute naming its parent, and this attribute is indexed by a clustering index. The method first performs a sequential scan of the parent file. 7 his yields a set of parent identifiers which are used as keys to the child file index, and a clustering index scan (CI) is then performed on the child file.

Appendix B. The Costs of Rival Retrieval Methods

B.l Sequential scan

First, the cost of a sequential scan must be computed. Following Yao and DeJong[YD78] we take the number of pages in the file cluster containing file F to be SCLUS(F). We know that

S(F)/PR(F)<= SCLUS(F) <= S(F)
where S(F) = # of records in F
PR(F)= # of records of F per page

We have already made the simplifying assumption that relations are either stored alone on pages (the "standard" case) or are "embedded" as the parent or child relation in a 1 to N relationship. A sequential scan will fetch every page in the file cluster. The number of pages fetched from secondary storage is the cost metric, so the total cost depends upon how thefile is stored. There are three cases:

Case 1. F is stored by itself on all its pages. In this case,

[SI) SCLUS(F) = S(F)/PR(F)

Case 2. F is involved in a parent-child relationship and is interleaved with F'. Let f be the parent-child fanout, and assume that there are no "childless" parents, no "orphan" children, and that every parent has exactly f children, Now,

SCLUS(F) = SCLUS(F') = (S(F) + S(F'))/PR

assuming PR(F) = PR(F') = PR. Now suppose F is the parent and F' is the child. Then,

$$S(F') = f * S(F)$$

Therefore,

[S2] $$SCLUS(F) = ((f+1) * S(F))/PR$$

Case 3. Now considering the child file F', we find

[S3] $$SCLUS(F') = ((1 + 1/f) * S(F'))/PR$$

Because the sequential scan fetches every page of the file cluster, the cost of the method is just the value of SCLUS in the appropriate case.

B.2 Indexed scans

The cost of a scan using an index is made up of two components: the cost of retrieving the index, and the cost of retrieving the data records using the index pointers.

a. Retrieving the index.
Let NLEV = # of levels in the index.
PI = # of index pointers per page (assumed to be a system constant).
RSEL = restriction selectivity (see section 6.2).

Then this cost is

[I1] $$NLEV + (S(F)*RSEL)/PI$$

because we need NLEV pages to get to the terminal level of the index, plus we will need pointers to S(F)*RSEL records, and there are PI of those per page.

b. Retrieve the records using the pointers.

Again following Yao and DeJong, the key parameter determining this cost is the average number of pages that must be retrieved to get from one record of a file to the next, denoted by PGS(F,F); PGS(F,G) is used in general when two different files are considered.

The cost of retrieving K records is K*PGS(F,F), and K = S(F)*RSEL. Therefore, the cost of record retrieval using index pointers is

[I2] $$S(F) * RSEL * PGS(F,F).$$

Expressions [I1] and [I2] can now be used to give the total cost of indexed retrieval in various cases of clustering/nonclustering indexes, standard/embedded storage, parent/child role, and size of fanout.

The expression giving the cost of indexed retrieval in any of these cases is

[13] NLEV + ((S(F)*RSEL)/PI) + (s(F) * RSEL * PGS(F,F))

The parameter PGS(F,F) will be affected by three conditions: whether or not the index is a clustering index, whether the relation is stored in the standard way or is embedded with another, and if it is embedded, the size of the fanout.

If the index is not a clustering index, then the pointers obtained from the index point to records throughout the file cluster in any order. This means that nearly every record requires fetching a new page independently of whether the data pages are stored in a "standard" or "embedded" way (see Appendix A). Therefore, we must assume

[14] PGS(F,F) = 1 if the index is not a clustering index

If the index is a clustering index, then pointers are obtained in the same sequence that the data records are stored, so the cost of retrieval depends upon whether or not the file is "embedded". In arty case, PGS(F,F) is inversely proportional to the number of records per page, PR(F). An example illustrates this. If there are 20 records per page and we want 200 records, then we must access approximately 10 pages; if there are 40 records per page, then about 5 pages are needed. (This could be slightly off because of straying over page boundaries.)

Now we consider different storage situations. First, suppose the standard case where F is alone in SCLUS(F). Then, as suggested above,

[15] PGS(F,F) = 1/PR(F) (approximately) for a clustering index and "standard" storage

Next, suppose that F is a parent relation and F' is a child relation in a 1 to N relationship, and they are stored together (the "embedded" case). If the fanout f (= number of children, N) is greater than PR (again assuming PR(F) = PR(F') = PR) then no more than one of the parent records of F can be stored on a single page. Therefore,

[16] If f >= PR then PGS(F,F) = 1 for parent file F

If the fanout is less than PR, we must have space for f+1 records to get from one parent record to the next. Therefore

[17] If f < PR then PGS(F,F) = (f+1)/PR (approximately) for parent file F

At the same time, any K consecutive records of the child relation F' require space for those K plus the appropriate number of parent records. Space is therefore needed for K * (1 + (1/f)) records. This means that

[18] PGS(F',F') = (1 + (1/f))/PR (approximately) for child file F'

Now that we have the formulas for the key parameter PGS(F,F) for various database structures, we can now give the corresponding complete cost formulas for indexed retrieval.

Method	Cost Formula
NCI	NLEV + ((1/PI) + 1) * (S(F) * RSEL)
CI, standard storage	NLEV + ((1/PI) + (1/PR)) * (S(F) * RSEL)
CI, embedded parent, f >= PR	NLEV + ((1/PI) + 1) * (S(F) * RSEL)
CI, embedded parent, f < PR	NLEV + ((1/PI) + ((f+1)/PR)) * (S(F) * RSEL)
CI, embedded child	NLEV + ((1/PI) + ((1+(1/f))/PR)) * (S(F) * RSEL)

B.3 Cost Formulas for Rival Methods

Using the expressions derived above for different components of the retrieval cost, we obtain the following formulas for the costs of different methods. These will be the basis for computing constraint thresholds.

[CI] Cost(SS) = S(F)/PR(F) where F is stored alone.

[C2] Cost(SS) = ((f+1) * S(F))/PR where F is an embedded parent and it is assumed that PR(F) = PR(F') = PR.

[C3] Cost(SS) = ((1 + 1/f) * S(F))/PR where F is an embedded child

[C4] Cost(NCI) = NLEV + ((1/PI) + 1) * (S(F) * RSEL)

[C5] Cost(CI) = NLEV + ((1/PI) + (1/PA)) * (S(F) * RSEL) where F is stored in the "standard" way.

[C6] Cost(CI) = NLEV + ((1/PI) + 1) * (S(F) * RSEL) where F is an embedded parent and f >= PR.

[C7] Cost(CI) = NLEV + ((1/PI) + ((ftl)/PR)) * (S(F) * RSEL) where F is an embedded parent and f < PR.

[C8] Cost(CI) = NLEV + ((l/PI) + ((1+(1/f))/PR)) * (S(F) * RSEL) where F is an embedded child.

[C9] Cost(LI) = Cost(SS') + Cost(CI), or Cost(SS') + Cost(NCI). [Here, SS' refers to the cost of a sequential scan of the separate file]

Appendix C. Constraint Thresholds for Rival Methods

Given the cost formulas calculated above, it is easy to compute the value for constraint selectivity, RSEL,

at which a scan that uses an index costs less than a sequential scan. The constraint threshold expressions will be given here for a comparison of sequential scan against each of the three rivals: clustering index scan (CI); nonclustering index scan (NCI); and linked index scan (LI). The comparisons will be made for three types of storage: fife F is stored alone on its data pages; F is an embedded parent; and F is an embedded child.

in ail the calculations, it is assumed that NLEV, the cost of descending the index tree once, is negligible. it is also assumed that the first relation searched by the Li method is stored alone on its data pages, and that it gives values into a clustering index.

We will also give typical values for the constraint threshold, using the following parameter values:

Pi = # of index pointers per page = 200 PR = # of data records per page = 100 q = PI/PR = 2 f = f' = fanout = 10 (except in embedded cases where f >= PR)

Two further parameters will be used to simplify the constraint threshold expressions:

w = q * (f+1) v = q * (1 + (l/f)).

Here are the constraint threshold figures. Numerals in brackets, such as [2], refer to notes at the end ot the table.

Rival method	Constraint threshold expression	Typical value
1. F is stored alone.		
CI	q/(1 + q)	.67
NCI	q/(1 + PI)	.01 [1]
LI	(1 - (1/f')) * (q / (1 + q))	.60
2. F is an embedded parent.		
CI (f >= PR)	v/(1 + PI)	1 [2]
CI (f < PR)	v/(1 + v)	.96 133
NCI (f >= PR)	v/(1 + PI)	1 [4]
NCI (f < PR)	v/(1 + PI)	.11 [4]
LI (f >= PR)	(w-(q/f'))/(1+PI)	1 [5]
LI (f < PR)	(w-(q/f'))/(1+v)	.95
3. F is an embedded chi id.		
CI	w/(1+w)	.69 [6]

NCI	w/(1+PI)	.01
LI	q/(1+w)	.63 [7]

Note 1. A nonclustering index turns out not to be very useful for the kind of subset extraction we **discuss in this paper. Its benefits become apparent only when a very small fraction of the** items in the **set are to be extracted.**

Note 2. it is not surprising that CI should always be favored here. When the fanout **exceeds the number of records per page, SS will always look at pages that contain only child records, no parent records,** while **CI will never look at those pages. For the typical value shown, recall that in this case, f >=** PR, so it is **roughly** 100, **not** 10.

Note 3. Again, CI is almost always favored. As the fanout **approaches** 1 **(one child per parent), the lower limit of the threshold is about .8, meaning that the work to retrieve the index begins to be** penalized.

Note 4. When f is greater than PR, the advantage of a clustering index over a nonclustering index disappears, because every new record will require fetching a new page no matter in what order the **pointers are obtained. When f is less than PR, the advantage of clustering reasserts itself.**

Note 5. We are dealing here with a two-level hierarchy; the fanout **from one file to the other** is f' and **the** fanout **within the second file is f. When f is close to PR, the threshold is actually slightly lower than 1, reflecting the added cost of the first scan.**

Note 6. We are almost back to the case where F is stored alone, as we have assumed that f = 10.

Acknowledgments

My thanks for thoughtful comments on this paper go to Sheldon Finkeistein, Jeffrey Ullman, and Gio **Wiederhoid at Stanford University, and Daniel Sagaiowicz and Earl Sacerdoti at SRI international. The research described here is part of the Knowledge Base Management System Project at Stanford** and SRI, **supported by the Advanced Research Projects Agency of the Department of Defense under contract** MDA903-77-C-0322.

Ref erences

[Ch78] **C. L. Chang,** *DEDUCE 2: Further Investigations of Deduction in Relational Databases* **in** [GM78], pp 20 l-236, 1978.

[GM78] **Herve Gallaire and Jack Minker (Eds.),** *Logic and* **Data Bases, Plenum Press, New York and London,** 1978.

[HS78] **Gary G. Hendrix, Earl D. Sacerdoti, et. al.,** *Developing a Natural Language interface to Complex Data,* **ACM Transactions on Database Systems, Vol. 3, No. 2, pp. 105-147, June** 1978.

[KK78] **Charles Kellog, Phillip Klahr, and Larry Travis,** *Deductive Planning and Pathfinding for Relational Data Bases,* **in [GM78], pp.** 179-200, 1978.

[Ki79] **Won Kim,** *Relational Database Systems,* **ACM Computing Surveys, Vol.** 11, **No.** 3, **pp.** 185-211, **September** 1979.

[Mc76] **Dennis J. McLeod,** *High Level Expression of Semantic Integrity Specifications in a Relational Data* **Base** *System,* **MIT Laboratory for Computer Science Technical Report 165, September** 1976.

[MM77] **James R. McSkimin and Jack Minker,** *The Use of a Semantic Network in a Deductive Question Answering System,* **Proceedings of the Fifth International Joint Conference on Artificial Intelligence, pp.** 50-58, 1977.

[Mo79] **Robert** *C.* **Moore,** *Handling Complex Queries in a Distributed Data Base,* **SRI International** Artiticial **Intelligence Center, Technical Note 170, October** 1979.

[NG78] **J. M. Nicolas and H. Gallaire,** *Data Base: Theory vs. Interpretation,* **in [GM78], pp. 33-54,** 1978.

[Ni71] **Nils J. Nilsson,** *Problem-Solving Methods in Artificial Intelligence,* **McGraw-Hill, New York,** 1971.

[Re78] **Raymond Reiter,** *Deductive Question-Answering on Relational Data Bases,* **in [GM78], pp.** 149-177, 1978.

[Sa77] **Daniel Sagalowict,** *IDA: An Intelligent Data Access Program,* **Proceedings of the Third International Conference on Very Large Data Bases, October** 1977.

[St753 Michael Stonebraker, *Implementation of Integrity Constraints and Views by Query Modification,* **Proceedings of the ACM SIGMOD International Conference on the Management of Data, pp 65-78, May 1975.**

[Ya79] S. **Bing Yao,** *Optimization of Query Evaluation Algorithms,* **ACM Transactions on Database Systems, Vol. 4,** No. **2, pp. 133-155, June 1979.**

[YD78] **S. Bing Yao and David De Jong,** *Evaluation of Access Paths in a Relational Database System,* **Purdue University Department of Computer Sciences Technical Report 280, August** 1978.