A System for Semantic Query Optimization

Sreekumar T Shenoy* and Z Meral Ozsoyoglu+

Computer Engineering and Science Department
and
Center for Automation and Intelligent Systems
Case Western Reserve University
Cleveland, Ohio 44106

ABSTRACT

This paper describes a scheme to utilize semantic integrity constraints in optimizing a user specified query The scheme uses a graph theoretic approach to identify redundant join clauses and redundant restriction clauses specified in a user query An algorithm is suggested to eliminate such redundant joins and avoid unnecessary restrictions In addition to these eliminations, the algorithm aims to introduce as many restrictions on indexed attributes as possible, thus yielding an equivalent, but potentially more profitable, form of the original query

1 Introduction

Semantic query optimization represents a relatively new approach for database query optimization Conventional query optimization is a well studied area in the database field Several query processing algorithms and heuristics have been proposed in the literature [Aho79, Astr76, Bern81, Blas77, Epst78, Gran80, Gotl75, Hevn79, Kim79, Maie83, Pale74, Ston76, Ullm82, Wong76, Yao79] Most of the major commercial database management systems make use of these techniques to some extent to answer the adhoc user queries Conventional optimization techniques mainly rely upon the syntactic knowledge and storage details The syntactic knowledge usually includes algebraic transformations and operator resequencing, whereas the storage details include availability of indices and clustering of storage The semantic query optimization, on the other hand, tries to exploit the knowledge about the relations domains of their instances, and various constraints associated with them

Even though there are earlier papers discussing semantic query simplification, this issue was formally introduced for the first time in [King81] and [HaZd80] The first paper analyzes the issue in an Artificial Intelligence context, and the second one views it from a database point of view Later, there was a substantial amount of research related to the theory and implementation of semantic rules for query optimization [ChFM84, ChGM86, Jark84, JaCV84, BrJa84 KuHa84, Morg84]

The work discussed here is most relevant to two of the above papers, [ChFM84] and [Jark84] [ChFm84] uses a scheme called semantic compilation to associate the set of valid and useful integrity constraints with each relations (or views) The motivation is to transform any queries on that relation (view) by using the associated constraints without searching the entire rule base In [Jark84] a graph approach is

Research partially supported by HSEQ Systems, Standard Oil Co, Cleveland, Ohio 44114

† Research partially supported by NSF under grant No DCR 8605554

described to integrate tableau techniques and syntactic simplification algorithms to optimize queries containing inequality restrictions Referential integrity constraints like key dependencies, functional dependencies, and value bounds are used to arrive at different forms of a given query The graph is used to unify attribute values based on referential constraints, to detect cycles that imply equal values for different attributes, and to predict queries with null answers

Both of the above schemes have certain limitations In [ChFM84], no clear way is suggested to categorize a given piece of semantic information as a rule or as a view Also, once the set of valid rules are identified for each relation (view), no method is described to select the profitable rules for a relation (view) in a query context Moreover, no mechanism is available to quantify the profitability of a rule for a relation in a query context We believe that evolution of a profitable query form should depend on three different factors, namely, relations, rules, and query, and this dependency should be dynamic Also, there should be well defined procedures to compare different query forms for their profitability In [Jark84], no scheme is available for an explicit representation of arbitrary semantic constraints The Prolog like view representation scheme allows to express a limited type of constraints on the variables appearing in view definitions In this method, it becomes the responsibility of the user to keep track of semantic constraints contained in a view definition Any changes in the constraints at a later stage makes maintenance and usage of these views difficult Also, since constraints are hardwired to the attributes of view definitions, they become unsharable by the similar attributes originating from the query

In this paper, we try to overcome the above difficulties by using explicit clausal representation [Kowa83] for integrity constraints as in [ChFM84], and by devising a mechanism for dynamic interaction between relations (views) and constraints in a given query context Among the valid constraints selected for interaction, only the profitable ones are finally used, and the profitability is decided by heuristics rules, global parameters, and some assumptions

This paper is organized as follows Section 2 describes various types of integrity constraints used in query transformation, their categorization, and representational details In section 3, heuristic rules adopted to determine the profitability of a transformation are discussed The heuristic rules are illustrated using an example database Section 4 deals with the query graph and its representation In section 5, semantic query transformation and various steps involved are discussed in detail Implementation details of semantic query optimization are described in section 6, including data structures, transformation algorithm, correctness and cost analysis of this process Section 7 concludes the paper

2. Integrity Constraints, their categories and representation

Semantic integrity constraints are laws, or expressions associated with the database that represent certain required properties of the data There are two broad classifications of integrity constraints, ie, state integrity constraints and transition integrity constraints [NiYa78] In this paper, we restrict our discussion to state integrity constraints The state integrity constraints can be further classified into conventional dependencies and semantic constraints Conventional dependencies include functional (and key) dependencies, value bounds, referential constraints etc A detailed discussion can be found in [Ullm82] Semantic constraints represent inter-relationships between chunks of data across the database relations In this paper, we do not address the issues of utilizing conventional dependencies for semantic optimization

In this work, we utilize two types of semantic integrity constraints, viz, implication integrity constraints and subset integrity constraints Clausal forms [Kowal83] are used to represent both types of integrity constraints

A clause is an expression of the form "A1, ,An --> B1, ,Bm" where A1, ,An,B1, ,Bm are atoms (or atomic formulae), n>=0 and m>=0 An atom (or atomic formula) is an expression of the form

"p(t1, ,tm)" where p is a m-place predicate symbol, t1, ,tm are terms, and m>=1 A term, in general, is a variable, a constant symbol or an expression of the form "f(t1, ,tm)" where f is a m-place function symbol, t1, ,tm are terms and m>=1 In our discussion, we consider only function-free terms

The atoms A1, ,An of the above clause are said to form the body of the clause, and B1, ,Bm are said to form its head In a clause, the negative atoms form its body and the positive ones form its head The atoms appearing in the body are the joint conditions of the clause, and the ones in its head are the alternative conclusions The conditions are sometimes referred to as antecedent atoms and the conclusions as consequent atoms A horn clause is defined as the one which has atmost a single atom in its head A horn clause with a null body and a non null head is called a unit clause

Definition 2 1 A subset integrity constraint is a superset-subset relationship between the domains of two different attributes of possibly two different relations

A subset integrity constraint is represented by a unit clause of the general form "--> r1 A1 ⊇ r2 A2" where r1, r2 are relations and A1, A2 are attributes This subset integrity constraint limits the set of values that r2 A2 can have to be a subset of the set of values that r1 A2 can have Note that this definition is not limited to the case of proper subsets

A classic example of subset integrity specification is "all managers are employees"

Definition 2.2 Implication integrity constraints define valid ranges of values (in terms of the comparison operators "=", "!=", ">=", and ">") that certain attributes of relations can take when some other attributes are restricted in the same or a different relation

An implication integrity constraint is represented by a horn clause An implication integrity constraint is said to be local if all its antecedent atoms refer to the same relation as its consequent atom does Otherwise it is called a cross constraint because, in such cases, the implication relates more than one relation The cross implication integrity constraints involve at least one join specification between relations

An implication integrity constraint can have either of the following two different forms
1) r1 A1 op1 k1 --> r1 A2 op2 k2
2) r1 A1 op1 k1,
r1 A2 op2 r2 A3 --> r2 A4 op3 k2
where r1, r2 are relations, k1, k2 are constants, and each of op1, op2, op3 is one of the comparison operators {=, !=, >=, >}

Examples of implication integrity constraints are "All employees in Sales department are over 35", "Only managers make more than 40K" etc

3. Heuristics and inference rules

Before formally describing the utilization of integrity constrains in semantic query transformation, we present a brief overview of various heuristic and inference rules used in semantic optimization The illustrations are based on an example database with the following relation schemes and integrity constraints

Example 3 1
Schema
supplier (Sname, Status, City)
material (Mname, Risk)
department (Dept, City, Manager)
shipment (Sname, Mname, Qty)
storage (Dept, Mname, Qty)
employee (Ename, Age, Sal, Dept)

Information on indices
shipment is indexed on Sname
department is indexed on City

Subset Constraints
material Mname is a superset of storage Mname

Implication Constraints and their clausal forms

Only d3 can store more than 450 tons of any material
storage Qty > 450 --> storage Dept = d3

All materials stored by d1 or d3 are of risk > 4
storage Dept = d1, storage Mname = material Mname --> material risk > 4

storage Dept = d3, storage Mname = material Mname --> material risk > 4

Benzene is always shipped in quantities > 500
shipment Mname = benzene --> shipment Qty > 500

Only supplier with status > 25 can ship in Qty > 400.
shipment Qty > 400, supplier Sname = shipment Sname --> supplier status > 25

Only department d1 has employees of age > 40
employee Age > 40 --> employee Dept = d1

Department d1 is located in London
department Dept = d1 --> department City = London

Only supplier s1 can ship materials in amounts exceeding any of their storage capacity
shipment Qty > storage Qty, shipment Mname = storage Mname --> shipment Sname = s1

We use four heuristic rules as suggested in [King81], namely, restriction elimination, index introduction, scan reduction, and join elimination The heuristic strategy of join introduction as in [King81] is not used in our approach In the following illustration we use a quel-like language [Ston76] for expressing queries

Restriction Elimination Remove a restriction from the query, if found redundant

Example 3.2.
Query Q1
(List all the shipments of Benzene in quantities > 300 tons)
Quel Form
retrieve (shipment all) where shipment Mname = benzene and shipment Qty > 300
Rule(s)
shipment Mname = benzene --> shipment Qty > 500
Query Q1'
retrieve (shipment all) where shipment Mname = benzene
Result
The unnecessary restriction on the attribute "Quantity" is eliminated

Index Introduction Introduce a restriction on an indexed attribute, if implied by the query

Example 3 3.
Query Q2
(List the managers in department d1)
Quel Form
retrieve (department Manager) where department Dept = d1
Rule(s)
department Dept = d1 --> department City = london
Relation "department" is indexed on attribute "City"
Query Q2'
retrieve (department Manager) where department Dept = d1 and department City = london
Result
A new constraint is obtained on indexed attribute "City" of the relation "department"

Scan Reduction Reduce the number of inner scans of the join by obtaining additional restrictions prior to the cross referencing operation

Example 3 4
Query Q3
(List the suppliers and their cities who ship in Qty > 450)
Quel Form
retrieve (supplier Sname, supplier City) where supplier Sname = shipment Sname and shipment Qty > 400
Rule(s)
shipment Qty > 400, supplier Sname = shipment Sname --> supplier status > 25
Query Q3'
retrieve (supplier Sname, supplier City) where supplier Sname = shipment Sname and shipment Qty > 400 and supplier status = 25
Result
Constraint on Status can be applied to the relation suppliers prior to cross matching step of

join between the relation "supplier" and "shipment" reducing the number of qualifying tuples from the relation "supplier" and hence the number of scans of the relation "shipment"

Join Elimination Eliminate a relation if it is joined to just another relation and none of its attributes contribute to the output

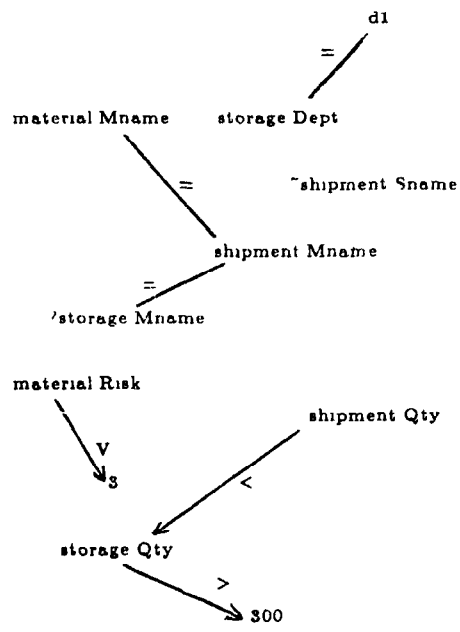
Example 3 5:
Query Q4
(List all the materials of risk > 3 stored in department d1)
Quel Form
retrieve (storage Mname) where storage Dept = d1 and storage Mname = material Mname and material Risk > 3
Rule(s)
storage Dept = d1, storage Mname = material Mname --> material Risk > 4
material Mname is a superset of storage Mname
Query Q4'
retrieve (storage Mname) where storage Dept = d1
Result
The join with the relation "material" is eliminated

4. Query graph and its representation

A query Q is a conjunction of join specifications of the form "r1 A1 op r2 A2" and the restriction specifications of the form "r1 A1 op k" where r1,r2 are relations, A1,A2 are attributes, k is a constant, and op is one of the comparison operators {=,'=,>=,>} The operators "<" and "<=" are not explicitly considered because a<=b and a<b are the same as b>=a and b>a respectively The answer of a query Q is the set of all tuples of the relations referenced in Q that satisfy Q, projected on the specified target attributes of Q

A query Q is represented by a query graph Gq which is a directed graph whose vertices are the attributes of the relations (attribute vertices) as well as the constants (constant vertices) involved in Q The edges of Gq are the join and restriction specifications in Q A join specification "r1 A1 op r2 A2" is represented by an edge from r1 A1 to r2 A2 with a label op Similarly, a restriction specification "r1 A1 op k" is represented by an edge from r1 A1 to the constant k with a label op While the "=" and "'=" edges are bidirectional, for the other edges the direction identifies the left and right operands of the label The edges representing a join specification are referred to as "join edges" whereas the ones denoting the restrictions are called "restriction (constant) edges"

Example 4 1.
Query Q5
(List the materials stored in d1 in more than 300 tons and risk > 3 that are shipped in quantities exceeding any of their storage capacity)
Quel Form
retrieve (storage Mname) where storage Dept = d1 and storage Qty > 300 and storage Mname = material Mname and material risk > 3 and shipment Qty > storage Qty and shipment Mname = storage Mname
Graph Gq
(Note that target attributes are identified by "?" and the indexed ones by "^")



Since a given query can have different equivalent forms, and hence different equivalent

query graphs, we adopt the notion of a condensed graph as a canonical representation of a query

The condensed graph Gc is derived from the query graph Gq by first grouping the attribute vertices of Gq into equivalence classes Any two vertices of Gq belong to the same equivalence class if they are connected by explicit equijoin edges or if there is a directed cycle of join edges incident with both the vertices A vertex forms an equivalence class by itself if it is not a part of any equijoin

The equivalence classes (as well as the constant vertices) of Gq are mapped as vertices in Gc as follows

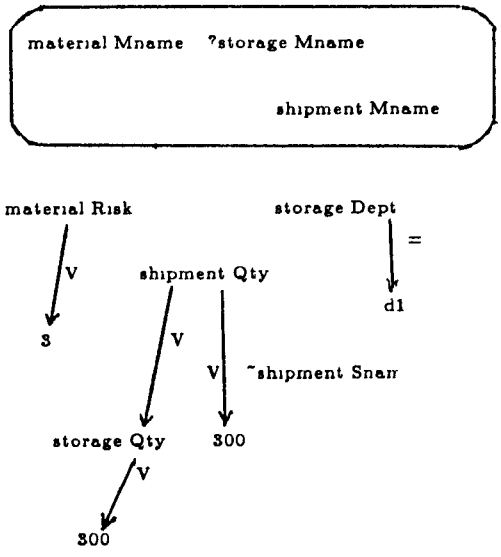
The non equijoin (!=,>=,>) edges of Gq are represented in Gc by their transitive reduction The transitive reduction [YuOz84] of a query graph with only join edges is a graph with the fewest number of join edges among all such query graphs having the same transitive closure The transitive reduction is obtained by first mapping all the non equijoin edges from Gq to Gc for the corresponding equivalence class vertices, removing all the redundant edges from Gc, and then replacing any multiple edges between two vertices by an equivalent single edge

The restriction (constant) edges of Gq are represented in Gc by their transitive closure over the transitive reduction of joins For this, first each restriction edge of Gq is mapped into Gc to restrict the vertex corresponding to the equivalence class Transitive closure of the mapped restriction edge is then obtained by explicitly representing the effect of that restriction, if any, on all the attribute vertices of the same connected component by adding new appropriate restriction edges to them This process of adding new restrictions for representing the transitive closure of a restriction is referred to as "propagation of a restriction through the join edges" When all the restriction edges are mapped from Gq to Gc, and propagated through the join edges, the graph Gc is said to have completed "query-implied expansion"

Example 4 2:

Graph Gc

(Condensed graph for the query graph Gq of the previous example)



Result(s)

1 The attributes material Mname, storage Mname, and shipment Mname form a single multimember node in Gc due to equijoins between them

2 The restriction edge "> 300" is propagated from "storage Qty" to "shipment Qty" through the join edge "storage Qty < shipment Qty"

Even though it is theoretically quite possible to represent the restrictions also by their transitive reduction, we select a transitive closure form for their representation for implementation reasons Representing the restrictions by their transitive closure over the join edges makes all the implied restrictions explicit These restrictions, when propagated through join edges and made more explicit, could restrict some indexed attributes of the query which, even though redundant, may be profitable if selections are performed before joins Moreover, such propagated restrictions could satisfy some new antecedent atoms of implication integrity constraints The corresponding consequent atom of those integrity constraints then get qualified to be added to the query

5. Query transformation

Semantic query transformation is the process of obtaining alternative query forms that are semantically equivalent to the original one The motivation of semantic optimization is to arrive at a more profitable query yielding the same answer, which could be syntactically different from the original query

In our approach semantic optimization of a query is obtained through three major stages, ie, query-implied expansion, semantic expansion, and semantic reduction

5 1 Query-Implied Expansion

The query-implied expansion, as described in the previous section, obtains a canonical representation Gc of the query through transitive reduction of joins and transitive closure of restrictions over joins This stage is independent of any semantic details and depends only on the query and the operator syntax Besides arriving at a canonical form of the query, this stage facilitates any early detection of contradictions in join or restriction specifications that could lead to a null answer

5 2 Semantic Expansion

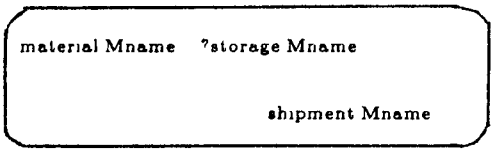
Semantic expansion iteratively adds any new restriction edges implied by the combination of query and semantic implication integrity constraints This is achieved by identifying the implication constraints whose antecedent atom(s) are satisfied by the query and adding the restriction corresponding to their consequent atom to the query From the original form, addition of each such restriction edges takes the query through various semantically equivalent forms till it reaches a stage Gm where no more new restrictions could be implied

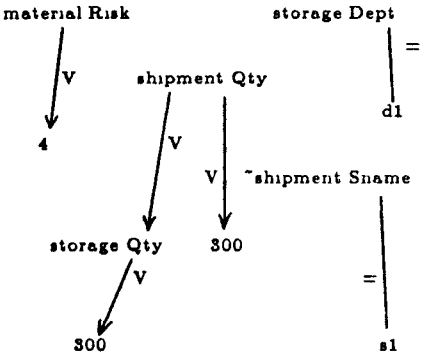
The purpose of semantic expansion is to incorporate any useful restrictions (possibly on indexed attributes) that are not present in the original query This assures that the query contains the maximal set of restriction edges that satisfy both the query and implication integrity constraints

Example 5 1:

Graph Gm

(Semantic expansion for the condensed graph Gc of the above example)





Result(s)

1 An additional restriction "shipment Sname = s1" is obtained where "shipment Sname" is an indexed attribute

2 The previous restriction "material Risk > 3" is replaced by a new restriction "material Risk > 4"

Rule(s)

1 shipment Qty > storage Qty, shipment Mname = storage Mname --> shipment Sname = s1

2 storage Dept = d1, storage Mname = material Mname --> material Risk > 4

5 3 Semantic Reduction

Semantic reduction detects all semantically redundant relations and restrictions present in the expanded form of the query and removes the non profitable ones

5 3.1 Relation Elimination

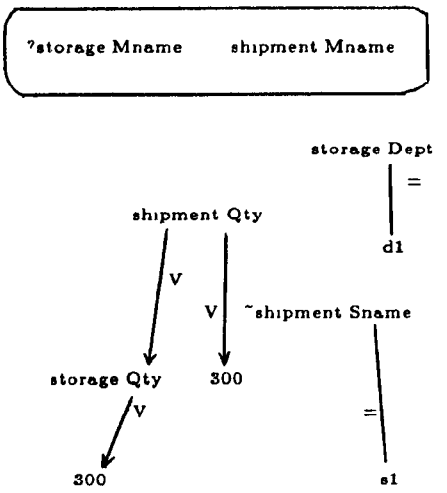
A relation is considered to be redundant if it becomes dangling so that none of its attributes or restrictions contribute to the answer Since the query graph is connected, elimination of a relation leads to the removal of its join to the rest of the query graph A relation elimination

is hence considered to be profitable because it eliminates the need of performing a join The graph, when all the redundant relations are removed from Gm, is denoted by Gr1

There are various conditions that a relation should satisfy to be classified as redundant First of all, it should be free from any target attributes of the query because a relation containing target attributes cannot be removed from a query Second condition is that all the restrictions on non-join attributes should be redundant By this, all such restrictions could be removed without altering the query semantics In this stage, the relation will have restrictions, if any, only on join attributes Third condition is that the relation should have at most one join vertex and the fourth one is that the relation does not have any non-equijoins Third and fourth conditions allow the transfer of all the restrictions on the only join attribute to the other side of the respective joins This makes the relation free from all restrictions Finally, there should be at least one other relation with a join attribute, say "S B", in the equivalence class containing the join attribute of this relation, say "R A", such that the subset dependency "R A S B" holds

More formally, a relation R is redundant if satisfies all of the following conditions
a) R is target-free
b) All the restriction edges on non-join vertices of R are redundant
c) R has at most one join attribute
d) R does not have any non-equijoins
e) There is at least one other relation with a join attribute, say "S B", in the join class containing the join attribute of R, say "R A", such that the subset dependency "R A $\subseteq$ S B" holds

Example 5 2
Graph Gr1
(Relation elimination for the expanded graph Gm of the above example)



Result(s)
The relation "material" is eliminated from the query
Rule(s)
1 storage Dept = d1, storage Mname = material Mname --> material Risk = 4
2 "material Mname" is a superset of "storage Mname"

5 3 2 Restriction (Qualification) Elimination

A restriction is redundant if it is satisfied by the consequent atom of an implication integrity constraint of which all the antecedent atoms are satisfied by the query The strategy of removing a redundant restriction from a relation largely depends on whether selections will be performed in that relation before joins This, in turn, depends on whether the join attribute in that relation is indexed or not If the relation has at least one indexed join attribute, it is assumed that the restrictions in that relation will be performed along with the join, and not before it This is because, with a given set of matching values for the join attribute, location of tuples becomes easy through the (indexed) join attribute and the restriction(s) could be checked during the same time On the other hand, if no join attribute of the relation is indexed, we assume that the selections in that relation will be performed before joins

In the cases where selections are performed before joins, locally redundant restrictions on indexed attributes become profitable provided none of the antecedent atoms of the

corresponding local implication constraint are on indexed attributes This is because, the redundant restriction introduces an indexed scan to replace the sequential scan of the relation Similarly, all the cross redundant restrictions become profitable if selections are performed before joins The reason is that such a restriction additionally limits the effective size of the relation before the join operation, thus resulting in a scan reduction A restriction, even though redundant, is considered to be profitable if it is on a join attribute since it may provide a better join strategy

The graph resulting from deleting all non profitable restrictions from Gr1 is denoted as Gr2 For the query graph Gr1 shown in the above example, no restriction is qualified for elimination That is, Gr2 is the same as Gr1 in this case

Unlike in the case of relation (join) elimination where all the redundant joins are removed, redundant restrictions are retained if they are found profitable But identifying a restriction to be profitable, as mentioned above, depends mainly on the estimation of the sequence of selections and joins This sequence, in reality, is determined by various factors outside the scope of this work, like relation sizes, optimizer statistics, optimizer intelligence, and sequence of specification of equality joins This might result in erroneous classification of profitable restrictions at times But by and large, this strategy provides a simple and reliable method to achieve the identification

5 4 Conversion from the condensed query graph

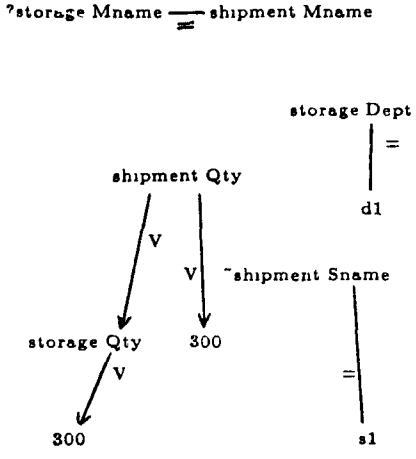
When the semantic expansion and reduction are completed, the query graph is converted back from its condensed form to the original one This is achieved by replacing each multimember node of Gr2 by any spanning tree on its attribute vertices connected by equijoin edges Any one attribute vertex, an indexed one if available, of the the multimember node is chosen for joining the spanning tree to other spanning trees or single attribute vertices Also, the restriction(s) on the multimember node are mapped as restriction(s) on all the attribute vertices This form of the graph, being the final one, is denoted by Gf

Note that, as in the case of restriction elimination, this strategy of graph conversion could also produce suboptimal results Cost of equijoins involving three or more attribute vertices depends on the their connecting spanning tree as well as the order in which they are considered for join Similarly, selecting an attribute vertex for joining the spanning tree to other vertices could also make a difference in cost For example, while selecting the edges of the spanning tree, priorities are given to the attributes of these relations which have one or more other joins between them In general, issues like relation sizes and selectivities should also be considered in selecting the spanning tree for multimember equivalence classes

Example 5.3:

Graph Gf

(Final form of the graph converted from Gr2 of the previous example)



Result(s)

The multimember node is replaced by a spanning tree

Quel Form

retrieve (storage Mname) where storage Dept = d1 and storage Qty > 300 and shipment Qty > storage Qty and shipment Qty > 300 and storage Mname = shipment Mname and shipment Sname = s1

Benefit(s)

Relation "material" eliminated from the query
Additional restriction obtained on indexed attribute shipment Sname
Additional restriction obtained on join attribute shipment Qty

6 Implementation details of query optimization

In this section we present the implementation specific details of the query optimization process This includes data structures for integrity constraints, algorithm for query transformation, and its correctness and cost analysis

6.1 Data Structures for integrity constraints

Four major data structures are used to represent integrity constraints and their status with respect to the query context These data structures are Cs (Subset Integrity Constraints), Ci (Implication Integrity Constraints), Lr (List of Restriction Atoms), and Lj (List of Join Atoms)

All the four structures, Cs, Ci, Lr, and Lj, can be represented in tabular forms Following is the list of fields for each of them

Subset Integrity Constraint, Cs
(Superset attribute,
Subset attribute)

Implication Integrity Constraints, Ci
(Constraint Id#,
Antecedent Atom,
Consequent Atom
Type,
Count *,
Flag *)

List of Restriction Atoms, Lr
(Restriction Atom,
Pointers,
Flag *)

List of Join Atoms, Lj
(Join Atom,
Pointers,
Flag *)

The "*" indicates that the field is sensitive to the query context Other fields derive their values only from the database semantics

The structure Cs contains the list of subset specifications The fields "superset" and "subset" define the subset dependencies between a pair of attributes of database relations

Ci lists all the implication integrity constraints Besides having fields for constraint identity #, antecedent atom, and consequent atom, it has three extra fields called "type", "count", and "flag" The field "type" indicates whether the integrity constraint is a local or cross constraint The field "count" represents the number of antecedent clauses in the implication constraint that are not satisfied by the query graph at any given stage of processing This count is initialized with the number of antecedent clauses present in the implication constraint A count 0 indicates that integrity constraint can be used The field "flag" indicates whether the constraint has already been used or not

Lr is a list of unique antecedent restriction atoms from all the implication integrity constraints The field "pointers" points to the set of integrity constraints in Ci that contain this as an antecedent atom The "flag" field indicates whether this restriction atom satisfies the query at any given stage of query transformation

The structure Lj, similar to Lr, contains a list of unique antecedent join atoms from the implication constraints

The four structures are illustrated below using the semantic details of the example database

Subset Integrity Constraint, Cs

Superset Attribute	Subset Attribute
material Mname	storage Mname

Implication Integrity Constraint Structure, C_i

#	Antecedent Atom(s)	Consequent Atom	Ty	Cnt	Flg
1	storage.Qty > 450	storage.dept = d3	L	1	
2	storage.Mname = material.Mname storage.Dept = d1	material.Risk > 4	C	2	
3	storage.Mname = material.Mname storage.Dept = d3	material.Risk > 4	C	2	
4	shipment.Mname = benzene	shipment.Qty > 500	L	1	
5	supplier.Sname = shipment.Sname shipment.Qty > 400	supplier.Status > 25	C	2	
6	employee.Age > 40	employee.Dept = d1	L	1	
7	department.dept = d1	department.City=london	L	1	
8	shipment.Qty > storage.Qty shipment.Mname = storage.Mname	shipment.Sname = s1	C	2	

List of Restriction Atoms, Lr

Restriction Clause	Ptrs	Flag
employee Age > 40	6	
department Dept = d1	7	
shipment Mname = benzene	4	
shipment Qty > 400	5	
storage Qty > 450	1	
storage Dept = d1	2	
storage Dept = d3	3	

List of Join Atoms, Lj

Join Clause	Ptr	Flg
storage Mname = material Mname	2,3	
supplier Sname= shipment Sname	5	
shipment Qty > storage Qty	8	
shipment Mname = storage Mname	8	

6.2 Algorithm for query transformation

/* Step 1 Obtain Query Implied Expansion Construct Gc from Gq */

a (map the vertex set) For each connected component c in Gq, partition the vertices into equivalence classes so that any two vertices are in the same equivalent class if they are connected by an explicit equijoin edge or if there is a directed join cycle with only ">=" edges incident at both of them A vertex forms an equivalence class by itself if it not a part of any equijoins These equivalence classes (as well as constants) of Gq are mapped as vertices of Gc

b (map the join edge set) For any two equivalence classes E_i,E_j, $i \neq j$, in Gq, let S be the set of edges between the vertices in E_i and the ones in E_j Let v_i,v_j be the vertices in Gc corresponding to E_i, E_j If S_{ij} is empty, then there is no edge between v_i and v_j in Gc If all the edges in S_{ij} have the same label ">=", ">" or "!=", then there is a single edge with the same label from v_i to v_j in GJq If S_{ij} contains edges with different labels then there is a single edge with ">" label between v_i and v_j (unless there is a contradiction)

c (remove the redundant edges) The edges in each connected component of Gc are successively examined in any order, and those implied by transitivity are removed

d (map the restrictions) Each restriction of Gq is mapped into Gc as restriction on the vertex corresponding to the equivalence class The effect of this restriction on all the attribute vertices of the same connected component of Gc, if any, is explicitly represented by adding new appropriate restriction edges to them Local redundancies or conflicts resulted at the attribute vertices between these propagated restrictions and any existing restrictions are detected and resolved

e (detect any equijoins implied by restrictions) If the mapped or propagated restrictions imply any new ">=" cycles in the connected component, merge those vertices into a single one, and go to step c

```
/* Step 2  Initialize the Data Structures */
Lj flag  = 0 ,
Lr flag  = 0 ,
Ci flag  = 0 ,
Ci count = # of clauses in antecedent set

for each j in Lj
   if j is satisfied by Gc then
      j flag = 1 ;
      for all i in Ci pointed to by j
         decrement i count ,

for each r in Lr
   if r is satisfied by Gc then
      r flag = 1 ,
      for all i in Ci pointed to by r
         decrement i count ,
```

/* Step 3 Obtain Semantic Expansion, Gm */
while there exists at least one i in Ci with flag = 0 and count = 0 do
  i flag = 1 ,
  add the restriction i consequent to Gc, /* As in the construction of Gc */
  Tr = set of restrictions implied in Gc by addition of i consequent ,
  Tj = set of joins implied in Gc by addition of i consequent ,

  for each t in Tr
    for all r in Lr where r contains t and r flag = 0
      r flag = 1 ,
      for all i in Ci pointed to by r
        decrement i count ,

  for each t in Tj
    for all j in Lj where j contains t and j flag = 0
      jc flag = 1 ,
      for all i in Ci pointed to by j
        decrement i count ,

/* Step 4 Assign Labels */
Label all the indexed (i) vertices
for each i in Ci where i flag = 1
  label corresponding local(lr) and cross(cr) redundant restriction edges of Gm

/* Step 5 Eliminate Redundant Relations to obtain Gr1 */
while there is a relation R that satisfies the following
  a) R is target-free
  b) All the restriction edges on nonjoin vertices of R are redundant
  c) R has at most one join attribute
  d) R does not have any non-equijoins
  e) There is at least one other relation with a join attribute, say "S B", in the join class containing the join attribute of R, say "R A", such that the subset dependency "R A⊆ S B" holds
eliminate R from the query graph

/* Step 6 Eliminate redundant restrictions, to obtain Gr2 */
remove all the redundant restrictions if they are not profitable
  a) A redundant restriction on a join attribute is profitable
  b) If no join attribute of the relation is indexed, then all the cross redundant restrictions in that relation are profitable
  c) If no join attribute of the relation is indexed, then all locally redundant restrictions in that relation are profitable only if they are on indexed attributes and the corresponding antecedent restrictions of the implication constraints are on non indexed attributes

/* Step 7 Expand multimember nodes of Gr2 to obtain Gf */
  Replace each multimember node of Gc by a spanning tree on its attribute vertices connected by equijoin edges While selecting vertices of the spanning tree, assign priorities for attributes of those relations which have one or more other joins between them

  Select any attribute vertex, an indexed one if available, of the multimember node for joining the spanning tree to other spanning trees or single attribute vertices

  Map the restriction(s) of the multimember node as the restriction(s) on all the attribute vertices

6 3 Correctness of the algorithm

Let the original form of the query be denoted by Qo, the expanded form on completion of the expansion stage of the algorithm by Qm, and the final transformed form by Qf

Theorem 1: The query forms Qo, Qm, and Qf are semantically equivalent

Proof, Qo <==> Qm

The semantic transformation from Qo to Qm takes place in the expansion stage due to the addition of restriction edges to Gc from the consequent atoms of the implication integrity constraints (Note that propagation of restrictions through the join edges or merging the vertices incident at equijoin cycles are not considered as semantic transformations) Addition of each such restriction edge to Gc can be

assumed to transform the graph to a new query form The transformation from Qo to Qm thus constitutes a chain, Qo --> Qo1 --> Qo2 --> --> Qm

Here, the difference between two consecutive forms Qoi and Qoi+1 is exactly one restriction edge, say ei Introduction of ei to Qoi is due to the presence of a set of edges Ei in Qi such that there exists an implication constraint Ei --> ei in the structure Ci So addition of ei to Qoi does not alter the semantics of Qoi In other words, Qoi and Qoi+1 are semantically equivalent Extending the argument for the entire chain of transformations, it can be seen that Qo and Qm are semantically equivalent

Proof, Qm <==> Qf

The transformation from Qm to Qf is accomplished in the elimination stage of the algorithm As above, let us assume that the transformation from Qm to Qf can be represented by a chain, say Qm -- Qm1 --> Qm2 --> --> Qf

The transformation from Qmi to Qmi+1 can be due to elimination of a relation (join) or removal of a restriction edge by the semantic reduction stage of the algorithm

All the relations and restrictions qualified for elimination are the ones found semantically redundant, and hence their removal does not alter the semantics of the query Hence we conclude that Qmi and Qmi+1 are semantically equivalent, implying the semantic equivalence of Qm and Qf

6 4 Cost comparison of query forms

Theorem 2. Cost(Qf) < Cost(Qo) provided the estimation of selection-join sequence is valid

If Qf is different from Qo, let this difference be represented by three components 1) Set of restriction edges Ef+ that are present in Qf but not in Qo 2) Set of restriction edges Eo+ that are present in Qo but not in Qf, 3) Set of relations Ro+ that are present in Qo but not in Qf

The edges in Ef+ are syntactically or semantically redundant since they have been added by the algorithm to the initial graph during query-implied expansion or semantic expansion The fact that they were not eliminated during the semantic reduction implies that they belong to the "profitable" category, provided the estimated selection-join sequence holds good In this context they represent an additional profit for Qf as compared to Qo

All the edges in Eo+ are also syntactically or semantically redundant because otherwise they would have been retained in Qf too The reason for their removal by the semantic reduction stage was that they were not found to be profitable In other words, the edges in Eo+ represent an elimination of non profitable part from the original query, if the estimations on selection-join sequence holds good

In short, as compared to Qo, Eo+ represents the edges lost whereas Ef+ represents edges gained by Qf The strategy of adding and eliminating the restriction always concentrates on adding profitable restrictions and removing non profitable ones Both these components thus represent profit provided the sequence estimation of selections and joins are valid

The set Ro+ represents a clear profit for Qf because Qf does not have the corresponding joins

To conclude, the cost advantage of Qf over Qo can be represented by C as C = a1*|Ef+| + a2*|Eo+| + a3*|Ro+|, where a1, a2, a3 are scaling factors to reflect the relative importance of components as well as validity of the assumption on selection-join sequence If these assumptions are valid, C represents a positive quantity, and in that case larger the sets Ef+, Eo+, Ro+ are, higher is the resulting cost advantage

7 Conclusion

In this paper we have proposed and described a scheme for utilizing semantic integrity constraints for optimizing a database query We have tried to quantify the factors that decide the profit of a query and illustrated how relations,

rules, and query can interact to arrive at an optimum query form The major contribution of the work is a scheme that dynamically selects from a large collection of rules only the profitable ones for a relation in a query context

An algorithm is introduced to transform the initial query to a semantically equivalent one The algorithm has its best performance if the estimations of selection-join sequences holds good in reality Certain major factors like elimination of redundant joins are independent of these assumptions, anyway Cases where a query can be answered just using semantic rules and the ones where query conditions and/or semantic constraints imply a null answer are also handled efficiently by the algorithm In other cases, semantic rules aid the query processing by generating useful additional constraints or by eliminating existing redundant constraints

We are currently studying some additional types of integrity constraints and optimization strategies to incorporate in the algorithm Usage of conventional integrity constraints like functional dependencies along with the semantic constraints requires further analysis Methods like introducing an additional join to the original query (join introduction) as an optimizing scheme [King81] also needs further investigation from an implementation point of view

REFERENCES

[Aho79] Aho A V , Sagiv Y , and Ullman J D , Equivalences among relational expressions, SIAM J of computing 8, pp 218-246, 1979

[Astr76] Astrahan M M et al , SYSTEM R A relational approach to database management, ACM TODS 1 2, June 1976

[Bern81] Bernstein P A et al , Query processing in systems for distributed databases (SDD-1), ACM TODS 6 4, pp 602-625, Dec 1981

[Blas77] Blasgen M W , and Eswaran K P , Storage access in relational databases, IBM systems journal 16, 4, 1977

[ChFM84] Chakravarthy U S , Fishman D H , and Minker J , Semantic query optimization in expert systems and database systems, EDS 1984, 326-341

[ChGM86] Chakravarthy U S , Grant J , and Minker J , Foundations of semantic query optimization for deductive databases,

[Epst78] Epstein R , Stonebraker M , and Wong E , Distributed query processing in database systems, Proc ACM SIGMOD conference, Austin, pp 169-180, June 1978

[Gran80] Grant J , and Minker J , Optimization in deductive and conventional database systems, pp195-234 in Advances in database theory, vol 1, ed Gallaire H , Minker J , and Nicolas J M , Plenum Press 1980

[Gotl75] Gotlieb L R , Computing joins of relations, ACM SIGMOD international symposium on management data, pp 55-63

[HaZd80] Hammer M , and Zdonik S B Jr , Knowledge based query processing, VLDB 1980, 137-147

[Hevn79] Hevner A R , and Yao S B , Query processing in distributed database systems, IEEE transactions on software engineering 5, 3, pp 177-187, May 1979

[Jark84] Jarke M , External semantic query simplification A graph-theoretic approach and its implementation in Prolog, EDS 1984, 467-482

[JaCV84] Jarke M , Clifford J , and Vassiliou Y , An optimizing Prolog front end to a relational query system, Proceedings of ACM SIGMOD conference, Boston 1984, 296-306

[Kim79] Kim W , relational database systems, ACM computing surveys 3, 11, pp 185-212, Sept 1979

[King81] King J J , QUIST A system for semantic query optimization in relational databases, VLDB 1981, 510-517

[KuHa84] Kung R , Hanson E , Ioannidis Y , Sellis T , Shapiro L , and Stonebraker M , Heuristic search in database systems, EDS 1984, 96-107

[Maie83] Maier D , The theory of relational databases, Computer Science Press, 1983

[Morg84] Morgenstern M , The role of constraints in databases, expert systems, and knowledge representation, EDS 1984, 207-223

[NiYa78] Nicholas J M , and Yazdanian K , Integrity checking in deductive databases, pp 325-346 in Logic and databases, ed Gallaire H , and Minker J , Plenum Press, 1982

[Pale74] Palermo F P , A database search problem Information systems COINS IV (J T TOU, ed), Plenum Press

[Ston76] Stonebraker M R , Wong E , Kreps P , and Held G , The design and implementation of INGRES, ACM TODS 1 3, pp 189-222, Sept 1976

[Ullm82] Ullman J D , Principles of database systems, 2nd edition, Computer Science press, 1978

[Wong76] Wong E and Youssefi K , Decomposition A strategy for query processing , ACM TODS 1 3, pp 223-241, Sept 1976

[Yao79] Yao S B , Optimization of query evaluation algorithms, ACM TODS 4 2, pp 133-155

[YuOz84] Yu T C , and Ozsoyoglu Z M , On determining tree query membership of a distributed query, INFOR Aug 1983 261-28