Constructing Inter-Relational Rules for Semantic Query Optimisation

Barry G.T. Lowden and Jerome Robinson
*Department of Computer Science, The University of Essex,
Wivenhoe Park, Colchester, CO4 3SQ, Essex,
United Kingdom
Telephone 0044 1206 872773 Fax 0044 1206 872788 Email lowdb@essex.ac.uk*

Abstract:
Semantic query optimisation is the process by which a user query is transformed into a set of alternative queries each of which returns the same answer as the original. The most efficient of these alternatives is then selected, for execution, using standard cost estimation techniques. The query transformation process is based on the use of semantic knowledge in the form of rules which are generated either during the query process itself or are constructed according to defined heuristics. Previous research has tended to focus on constructing rules applicable to single relations and does not take advantage of the additional semantic knowledge, inherent in most databases, associated with relational joins. Our paper seeks to address this weakness by showing how the rule derivation process may be extended to the generation of inter-relational rules leading to significant improvements in the optimisation process.

Keywords: Intelligent Information Systems, Semantic Query Optimisation, Rule Generation, Inductive Learning.

1. Introduction

Semantic query optimization is the process of transforming a query into an alternative query that is semantically equivalent but executes in a shorter time [8, 4, 5, 14, 21]. A semantically equivalent query returns the same result set as the original query and is generated from it by the application of rules. SQO differs, therefore, from conventional optimisation in that it uses semantic information rather than the statistical data held within the DBM system catalogue [1, 3, 11].

Rules can consist of 'integrity constraints' that hold for all states of the database or 'dynamic rules' that hold only for a given state. Generally, rules can be supplied by the database engineer or derived automatically. Automatic rule generation methods include heuristic based systems [20], logic based systems [4], graph based systems [19] and data driven systems [9, 18].

A weakness of these earlier approaches is that they do not take advantage of the semantic knowledge inherent in join relationships and, as such, are limited to the generation of rules associated with a single relation only. Siegel gives limited attention to the problem but does not address the general case.

As an example consider an employee relation:

E (name, department, salary, status, benefit)

for which the following rules apply:

(i) *department = 'computing'* $\rightarrow$ *benefit = 'car'*

(ii) *benefit = 'car'* $\wedge$ *salary > 30K* $\rightarrow$ *status = 'executive'*

and the relation is indexed on *status*.

A query of the form Q*:'retrieve names of all staff in computing who earn more than* 35K' may thus be transformed, using the above rules, into Q': *'retrieve names of all executives who work in computing and earn more than 35K'.* If executives represent only 5% of the workforce then it may be seen that this semantically equivalent query will execute in around 5% of the time taken by the original since, as *status* is indexed, only 'executive' tuples need be checked for the two original conditions.

Clearly this is a significant saving, however assume further that there also exists a department relation:

D (dept-name, budget, employee-count, executive-count) indexed on *dept-name*

and an inter-relational rule:

(iii) E.status = 'executive' $\wedge$ *E.department = D.dept-name* $\rightarrow$ *D.executive-count > 0*

Using this additional information, our original query may then be transformed into Q'': *'retrieve names of all executives who work in computing and earn more than 35K' provided that the corresponding 'computing' tuple in relation D has a positive executive count'.*

A single access to D will determine whether this last condition is true or false and, if false, Q'' will return a NIL answer to the user without further processing thus reducing execution costs to a minimum. If the condition is true then the query will execute as Q'.

In general, given a query Q with conditions (C_1, …..C_k) and a rule set R, query transformation may be achieved by repeated use of the following reformulation operators:

(a) Addition – given a rule in R (x $\rightarrow$ y), if a subset of conditions in Q implies x, then we can add condition y to Q.
(b) Deletion – given a rule in R (x $\rightarrow$ y), if a subset of conditions in Q implies x, and y implies C_i, then we can delete C_i from Q.
(c) Refutation – given a rule in R (x $\rightarrow$ y), if a subset of conditions in Q implies x, and y implies $\neg C_i$, then we can assert that Q will return NIL.

In this paper we present a rule generation approach extended to include construction of inter-relational rules. The method initially formulates a low cost alternative query using inductive learning [2, 7] and then derives the rules needed to effect the transformation from the input query to the alternative.

The structure of the paper is as follows. In the next section we describe how the low cost alternative query is constructed and then show, in section 3, how this method may be extended to the inclusion of join conditions. Section 4 presents a rule derivation algorithm based on the equivalence of the input and optimum queries using a range of examples. Finally in section 5 we summarise our work and give conclusions.

2. Constructing the Low Cost Alternative

To illustrate the underlying approach, we begin with a simple example based on a single relation. Consider the SALES database shown in Figure 1 together with an SQL-like user query of the form:

Q: *select * from salesemp where salary > 30K;*

The answer set of this query is referred to as the set of *positive* instances which is shown in italics. Instances, which are not in the answer set, are referred to as *negative* instances. An alternative query which is semantically equivalent to the input query must therefore return all the positive instances and no negative instances. With this basic requirement in place our method seeks to construct a cheaper alternative by evaluating conditions, on all attributes of the relation, which hold for the answer set.

Salesemp (emp#, status, Ksalary, area-code, location (10 char))

emp#	status	Ksalary	area-code	location (10 char)
E100	mgr	20	25	London
E105	mgr	22	35	Hull
E109	rep	15	24	Leeds
E115	***exe***	***40***	***33***	***Glasgow***
E120	***exe***	***35***	***34***	***London***
E122	***exe***	***37***	***23***	***Cardiff***
E125	***exe***	***42***	***25***	***Leicester***
E130	exe	28	36	Leeds
E150	sup	18	45	London
E155	sup	17	33	Leeds
E160	rep	16	22	Hull
E166	sup	19	36	Slough
E172	rep	14	27	Windsor
E180	mgr	24	31	Hull
E190	mgr	21	32	Leeds

Figure 1. SALES Database

In the example, these conditions are:

emp# in (E115, E120, E122, E125)
status = ‘exe(cutive)’
area-code >= 23 and <= 34
location in (Glasgow, London, Cardiff, Leicester)

together with the original query condition:

salary > 30K

If *status* were an indexed attribute then intuitively we see that costs could be reduced by adding the condition *status = 'exe'* to the original query and evaluating only executive tuples for the salary condition.

Specifically, we construct a condition cost table as shown in Figure 2. Entries are calculated by evaluating the product of (comparison cost of each condition C) with the (number of tuples to be compared T). For our example environment, the comparison cost is deemed to be 'string length' when applied to strings and '2' when applied to integers. The number of tuples to be compared will depend on whether the attribute is indexed and, for the purposes of our example, we assume that *status* and *area-code* are indexed.

The *gain* of each condition is measured by the number of negative instances which it eliminates, as represented by the column G, and the *effectiveness* E of each condition is computed as G/C*T. Original conditions of the input query are included in the computation.

The values shown in the table serve only to illustrate the principles of the method, realistic figures will of course depend on the specific machine environment and take into account the associated access timings and mechanisms. For example it is likely that indexing would, in practice, require only one condition comparison thus increasing the value of E for that entry.

Candidate Conditions	**C**	**T**	**G**	**E**
emp# in (E115, E120, E122, E125)	*16*	*15*	*11*	*0.0458*
status = 'exe'	*3*	*5*	*10*	*0.6667*
area-code >= 23 and <= 34	*4*	*8*	*4*	*0.1250*
location in (Glasgow, London, Cardiff, Leicester)	*40*	*11*	*9*	*0.0205*
salary > 30	*2*	*11*	*11*	*0.5000*

Figure 2. Condition Cost Table

The algorithm for constructing the alternative query then proceeds as follows:

Algorithm A.

1. Let the alternative query condition list L := NULL;
2. Let the number of negative instances be N;
3. Evaluate conditions on all attributes which hold for the answer set (including original input query conditions);
4. Let x = the condition with highest E value;
5. Add x to L and subtract G(x) from N;
6. If N = 0 then return condition list L else recompute table, with respect to L, and go back to step 3.

In our example the condition *status = 'exe'* is added to the condition list L and the value of N is set to 1. The table is then recomputed with respect to the tuples identified by the conditions in L, as illustrated in Figure 3.

Candidate Conditions	**C**	**T**	**G**	**E**
emp# in (E115, E120, E122, E125)	*16*	*5*	*1*	*0.0125*
area-code >= 23 and <= 34	*4*	*4*	*1*	*0.0625*
location in (Glasgow, London, Cardiff, Leicester)	*40*	*5*	*1*	*0.0050*
salary > 30	*2*	*5*	*1*	*0.1000*

Figure 3. Revised Condition Cost Table.

At this stage, the highest E value is associated with the input condition *salary > 30* which is added to L. The value of N becomes zero, hence L is returned as the condition list of the low cost alternative query:

Q' : *select * from salesemp where status = 'exe' and salary > 30;*

which evaluates the salary condition on the 5 ' executive' tuples as opposed to the original input query which necessitated the retrieval of all 15 tuples in the database.

Clearly it is not advantageous to use every user input query for low cost query construction and selection will normally be limited to those classed as the more expensive to execute. This will increase the probability of the derived rules being useful for optimisation. Inclusion of the original input conditions in the evaluation process ensures that an alternative semantically equivalent query is always produced even though, trivially, it may be syntactically the same as the input query when a cheaper alternative cannot be found.

3. Inter-relational (Join) Constraints

We will first address the situation where an input query already includes a join condition and then examine the possibility of introducing further join conditions in order to generate a semantic equivalent.

Any well found user query, defined on more than one relation in a database, will reference, in the condition list, at least one attribute from each relation defined. Where there is more than one relation referred to in the output list then our system first decomposes the input query into a set of sub-queries comprising exactly one output relation and its associated condition list. In what follows we assume, therefore, that all input queries are of this form and also return a non-NULL unique answer set.

Let our SALES database schema now be extended to that of Figure 4 where join attributes are marked with an asterisk. It is possible to identify meaningful join attributes by their inclusion in user queries or, in some cases the system catalogue.

*salesemp (emp#, status, salary,*area-code,*location-code)*
*placement (main-town, *location-code, population)*
*contacts (name, company, credit-rating, *area-code, address, position, commission)*

Figure 4. Extended Relational Schema

A possible query on this extended database is:

Q1: *select emp#, company from salesemp, contacts where salary > 30 and salesemp.area-code = contacts.area-code and credit-rating > 5;*

which decomposes into two (single output relation) queries:

Q11*: select emp# from salesemp, contacts where salary > 30 and salesemp.area-code = contacts.area-code ;*

and

Q12: *select company from salesemp, contacts where credit-rating > 5 and salesemp.area-code = contacts.area-code*

each of which is strongly related to Q1 but has wider applicability in terms of subsequent rule derivation.

Taking Q11 as our example and assuming the same answer set as our earlier query Q, we see that the set of candidate conditions shown in Figure 2 will be extended to include the inter-relational condition:

salesemp.area-code = contacts.area-code.

Condition costs may be evaluated as previously discussed and the number of tuples accessed will depend on whether either or both of the join attributes are indexed. If neither attribute is indexed then the value of T will relate to the product of the cardinalities of the joining relations. In this particular example we note that *salesemp.area-code* is indexed hence the number of tuples accessed will be proportional to the cardinality of the relation *contacts*. If the cardinality of *salesemp* is 50K and that of *contacts* is 5K and the range of values of *contacts.area-code* defined by the answer set of query Q11 may be associated with 10K tuples in *salesemp,* then the gain achieved by condition *salesemp.area-code = contacts.area-code* is 40K.

Inclusion of an inter-relational condition, in a partially constructed alternative query condition list L, permits the addition of candidate conditions on the set of attributes, associated with the joining relation, to the search space. These are then eligible for inclusion in L. In our example, if the join condition:

salesemp.area-code = contacts.area-code

were selected then conditions on each attribute in *contacts,* matching the answer set, would be evaluated and added to the candidate condition list. Where this leads to the inclusion, in the alternative condition list, of new join attributes linking additional relations then we may widen the search space for candidate conditions even further.

Given information about meaningful join attributes, the above strategy can be extended to include the construction of inter-relational conditions not referred to explicitly in the input query.

Note that, in the foregoing, the search space of candidate conditions is only extended to a further level if the related join condition is selected for inclusion in the partially completed alternative condition list ie. it is sufficiently cost effective. Where, however, a joined relation contains indexed attributes then candidate conditions are constructed on these attributes even if the join condition is not included in the alternative condition list. This simple heuristic ensures that potentially beneficial indexed conditions are not excluded from the evaluation process.

Expanding the search space of candidate conditions, in this way, is effected by ensuring that the above heuristic is included in step 3 of Algorithm A and also modifying the recompute function in step 6 to include:

> If x is a join condition on a new relation S then construct conditions on the attributes of S and include them in the set of candidate conditions.

4. Rule Generation

The previous section has described the process for constructing an alternative query Q' which is semantically equivalent to the input query Q and cheaper to execute. We now use that equivalence as the basis for deriving the rules which would be needed to transform Q into Q'. The algorithm is shown in Figure 5.

Algorithm B:

```
Assume Q ≡ Q' (where Q = input query, Q' = cheaper alternative query);
Let the derived rule set R := NULL;
Let A be the output relation in the select clause of Q';
Let $C_1$ … $C_n$ be the non-join conditions in the where clause of Q';
For each C
        Let consequent of new rule r = C;
        Construct P where P = shortest join path from C to A;
        Let antecedent of r = (condition list of Q) UNION (join conditions in P);
        If r not trivial then add r to R;
```

Figure 5. Rule Generation Algorithm.

In essence, for each non-join condition in the alternative query, Algorithm B determines the shortest join path to the output relation. New rules are constructed with the non-join condition becoming the consequent and the antecedent being formed from the UNION of the join path and the condition list of the input query. As before we will illustrate our approach by means of an example.

Assuming our previous extended database schema of Figure 4, let the input query be:

Q: select emp# from salesemp, contacts, location
where salesemp.area-code = contacts.area-code and
salesemp.location-code = placement.code and
population > 40K and
commission > 0.5 and position = manager;

and our alternative cheaper equivalent be:

Q': select emp# from salesemp, contacts
where salesemp.area-code = contacts.area-code and
credit-rating > 5;

The alternative query has a single non-join condition (*credit-rating > 5)* in the where clause and the shortest join path is (*salesemp.area-code = contacts.area-code).*

The derived rule (removing duplicate conditions) is therefore:

R1: (salesemp.area-code = contacts.area-code $\wedge$ *salesemp.location-code = placement.code* $\wedge$
population > 40K $\wedge$ *commission > 0.5* $\wedge$ *position = 'manager')*
$\rightarrow$ *(credit-rating >5)*

Clearly, however, rules which contain a large number of antecedent conditions tend to be very specific to the input query. The next stage of the process, therefore, attempts to decompose the rule into a set of shorter sub-rules more useful for optimising a wider range of user queries. This is achieved by identifying subsets of antecedent conditions which:

(a) maintain rule *cohesion*. A rule is cohesive if there is a connecting relational path through all antecedent and consequent conditions.
    and where the condition subset
(b) still implies the consequent of the derived rule (by reference to the database).

The decomposition process considers each combination of the antecedent conditions and discards those which do not conform to the above criteria. Taking R1 as an example, the following resultant sub-rule is not cohesive:

(salesemp.area-code = contacts.area-code $\wedge$ *population > 40K)* $\rightarrow$ *(credit-rating >5)*

since there is no path connecting either *salesemp* or *contacts* to *population.*

On the other hand the sub-rule:

(salesemp.area-code = contacts.area-code $\wedge$ *commission > 0.5)* $\rightarrow$ *(credit-rating > 5)*

is cohesive and will be checked against the database for validity.

All valid rules are eligible for inclusion in the rule base although some may be more effective than others in optimising queries. Also rule bases derived in this way can rapidly become very large and, as such, may degrade the efficiency of the optimisation process, a problem referred to as the *utility problem* [6, 12, 16, 22].

Our system addresses these issues by subjecting derived rules to statistical (Chi-square) analysis [10] which identifies and ranks rules according to their effectiveness. The rule set can be thus maintained at an optimum size and consists of only those rules which have wide applicability to the query space.

Rule validity may also be affected by changes to the database itself leading to potential inconsistent or weakened rules. Such rules are identified in our system by a process of rule maintenance [17, 13, 15], and subjected to statistical re-evaluation and ranking.

5. Summary and Conclusions

We have described a rule derivation approach which includes the generation of inter-relational rules based on the semantic information inherent in relational joins. Such rules typically have long antecedents and a technique for rule decomposition has been presented to generate sub-rules which have more general applicability to the optimisation process.

Simulation experiments show that the method performs well and identifies all worthwhile inter-relational rules. We now intend to carry out more extensive field trials based on large publicly available datasets extracted from the University of Essex Data Archive.

References

[1] M.W. Blasgen and K.P. Eswaran. ‘*Storage and access in relational databases’*, IBM Systems Journal, 16(4), 363-377, 1977.

[2] Y. Cai, N. Cerone and J. Han, *‘Learning in relational databases: an attribute-oriented approach’*, J. Computational Intelligence, 7(3), 119-132, 1991.

[3] A.F. Cardenas. *‘Analysis and performance of inverted data base structures’*, Communications of the ACM, 18(5), 253-263, 1975.

[4] S. Chakravarthy, J. Grant and J. Minker, *‘Logic-based approach to semantic query optimisation’*, ACM on Database Systems, 15(2), 162-207, 1990.

[5] G. Graefe and D. Dewitt, ‘*The EXODUS optimiser generator*’, Proc. ACM-SIGMOD Conf. on Management of Data, 160-171, May 1987.

[6] J. Han, Y. Cai and N. Cercone, ‘*Data-driven discovery of quantitative rules in relational databases*’, IEEE on Knowledge and Data Eng., 5(1), 29-40, 1993.

[7] D. Haussler, *‘Quantifying inductive bias: AI learning algorithms and Valiant’s learning framework’*, J. Artificial Intelligence, 36, 177-221, 1988.

[8] J. King, QUIST: *‘A system for semantic query optimisation in relational databases’*, Proc. 7th VLDB Conf., 1981.

[9] B.G.T. Lowden, J. Robinson and K.Y. Lim, *‘A semantic query optimiser using automatic rule derivation’*, Proc. Fifth Annual Workshop on Information Technologies and Systems, Netherlands, 68-76, December 1995.

[10] B.G.T. Lowden and J. Robinson, *'A statistical approach to rule selection in semantic query optimisation'.* Proc. 11th International Symposium on Methodologies for Intelligent Systems, LNCS Springer Verlag, 330-339, Warsaw, June 1999.

[11] L. F. Mackert and G. M. Lohman, '*R* optimizer validation and performance evaluation for local queries*', ACM-SIGMOD, 84-95, 1986.

[12] G.Piatetsky-Shapiro and C. Matheus, *'Measuring data dependencies in large databases*', Knowledge Discovery in Databases Workshop, 162-173, 1993.

[13] J. Robinson and B.G.T. Lowden, '*Data analysis for query processing'*, Proc. 2nd International Symposium on Intelligent Data Analysis, London, 1997.

[14] J. Robinson and B.G.T. Lowden, *'Semantic optimisation and rule graphs'*, Proc. 5th KRDB Workshop, Seattle, WA, May 1998.

[15] J. Robinson and B.G.T. Lowden, *'Distributing the derivation and maintenance of subset descriptor rules'*, Proc. 7th International Conference on Information Systems and Synthesis, Orlando USA, July 2001.

[16] I. Savnik and P.A. Flach, *'Bottom-up induction of functional dependencies from relations'*, Proc. Knowledge Discovery in Databases Workshop, 174-185, 1993.

[17] A. Sayli and B.G.T. Lowden, *'Ensuring rule consistency in the presence of DB updates'*, Proc. XII International Symposium on Computer & Information Sciences, Turkey, October 1997.

[18] S. Shekhar, B. Hamidzadeh and A. Kohli. '*Learning transformation rules for semantic query optimisation: a data-driven approach*', IEEE, 949-964, 1993.

[19] S.T. Shenoy and Z.M. Ozsoyoglu, '*Design and implementation of semantic query optimiser*', IEEE Transactions on Knowledge and Data Eng., 1(3), 344-361, 1989.

[20] M. Siegel, E. Sciore and S. Salveter, *'A method for automatic rule derivation to support semantic query optimisation*', ACM on Database Sys., 17(4), 563-600, 1992.

[21] Yu C. and Sun W., '*Automatic knowledge acquisition and maintenance for semantic query optimisation'*, IEEE Transactions on Knowledge and Data Engineering, Vol. 1, No. 3, 362-375, 1989.

[22] W. Ziarko, *'The discovery, analysis and representation of data dependencies in databases'*, Knowledge Discovery in Databases, AAAI Press, 195-209, 1991.