Automatic Relationship Discovery in Self-Managing Database Systems

Ihab Ilyas[1] Volker Markl[2] Peter J. Haas[2] Paul G. Brown[2] Ashraf Aboulnaga[2]
[1]Purdue University, West Lafayette, Indiana
[2]IBM Almaden Research Center, San Jose, California
ilyas@cs.purdue.edu, {marklv,phaas,pbrown1,aashraf}@us.ibm.com

1. Introduction

Relational database management systems rely on the query-optimizer component to choose a minimum-cost plan for accessing the data. The optimizer's cost estimates rest on "selectivity" estimates, that is, estimates of the amount of data that will be accessed at each stage of processing. Such selectivity estimates are in turn based on a set of summary statistics about the data (i.e., about the relational tables and columns) that are maintained in the system catalog. In traditional systems, the user specifies the set of catalog statistics to collect and maintain. This can be a daunting task, because the number of potential statistics, especially multivariate statistics on groups of columns, becomes huge as the number of tables and columns increases. The user therefore must somehow identify the "most important" set of statistics. A key goal of recent research in self-managing database systems is how to automate this task, i.e., how to quickly and efficiently discover the most important statistical dependencies between columns without requiring intervention by a highly skilled database administrator.

One approach to this problem is to use feedback from user queries. LEO, the DB2 learning optimizer [2], is a typical example of this approach. By comparing the actual results with optimizer estimates, LEO can detect errors caused by inaccurate statistics and missed correlations and create adjustment factors that can be applied in the future to improve the optimizer's estimates. The advantage of query-driven approaches is that they scale well and are efficient, because they focus their efforts on columns that appear in actual queries. The disadvantage is that the system can produce poor estimates—and hence choose poor plans—if it has not yet received enough feedback, either during the initial start-up period or after a sudden change in the workload.

In this paper we describe CORDS [1], an algorithm that automatically discovers correlations and soft functional dependencies (FDs) between pairs of columns and, based on these relationships, determines a set of statistics to maintain. This data-driven technology is an essential complement to query-driven approaches such as LEO, helping to ensure acceptable performance during slow learning periods. CORDS focuses on column pairs because this greatly simplifies the algorithms, and experiments have shown that the marginal benefit of capturing n-way dependencies for $n > 2$ is relatively small. By "correlations," we mean general statistical dependencies, not merely approximate linear relationships as measured, for example, by the classical Pearson correlation coefficient. By a *soft* FD between columns C_1 and C_2, we mean a generalization of the classical notion of a "hard" FD in which the value of C_1 completely determines the value of C_2; see [1] and references therein. In a soft FD (denoted by $C_1 \Rightarrow C_2$), the value of C_1 determines the value of C_2 not with certainty, but merely with high probability. The column pairs examined by CORDS may be in the same table or different tables.

CORDS first searches for column pairs that are likely to be related in an interesting and useful way by systematically enumerating candidate pairs and simultaneously pruning unpromising candidates using a flexible set of heuristics. For each surviving candidate pair, a statistical analysis is applied to a sample of column values in order to identify correlations and soft FDs. CORDS then recommends to the optimizer sets of multivariate "column group" (CG) statistics to maintain. Use of these statistics can lead to much more accurate selectivity estimates.

In developing CORDS, we have found that algorithmic simplicity and judicious use of sampling can lead to efficient and highly scalable self-management algorithms that are suitable for immediate incorporation into commercial systems. Our work also shows the importance of proactive behavior, rather than merely feedback-driven reactive behavior, in a self-managing system. The CORDS technology is potentially applicable to other autonomic systems in which automatic detection of correlated data—such as event or transaction data—is useful. The following sections describe CORDS in more detail; see [1] for a discussion of related work.

2. Candidate Generation

A *candidate* in CORDS is a triple (a_1, a_2, P), where a_i $(i = 1, 2)$ is a column of the form $R.c$, such as `CARS.Make` or `SALES.City`. The quantity P is a "pairing rule" that specifies which particular a_1 values get paired with which particular a_2 values to form the set of potentially interesting column-value pairs. When the columns lie in the same table R and each a_1 value is paired with the a_2 value in the same row, the pairing rule is called *trivial*. CORDS also allows columns a_1 and a_2 to lie in different tables, say R and S, where R and S might reasonably be joined during query processing. A pairing rule P in this case is simply a two-table join predicate between R and S, such as `R.custno = S.custno`.

CORDS first generates all candidates having a trivial pairing rule. CORDS then finds all nontrivial pairing rules that "look like" a key-to-foreign-key join predicate, since such join predicates are likely to occur in query workloads. Each nontrivial pairing rule P connects a pair of tables R and S, and CORDS generates all candidates of the form $(R.a, S.b, P)$. To find the nontrivial pairing rules, CORDS first identifies the set K comprising columns that are either declared primary or unique keys, together with each "undeclared key," that is, each column a not of these two types such that #distinctValues(a)/#rows$(a) \geq 1 - \epsilon_1$. (Here ϵ_1 is a param-

Proceedings of the International Conference on Autonomic Computing (ICAC'04)
0-7695-2114-2/04 $20.00 © 2004 IEEE

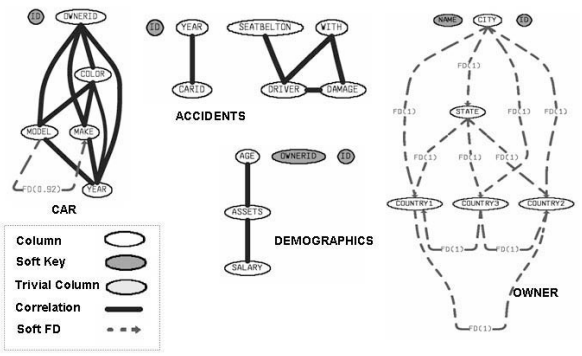

Figure 1. Dependency graph for the *Accidents* database.

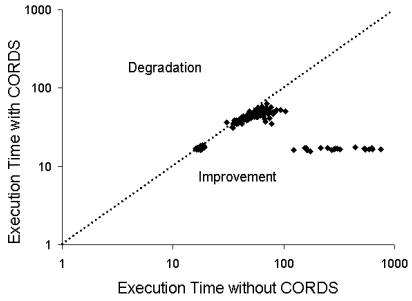

Figure 2. Effect of CORDS on performance.

eter of the algorithm.) For each column $a \in K$, CORDS examines every other column in the schema to find potential matches. A column b is considered a match for column a if either (1) column a is a declared primary key and column b is a declared foreign key for the primary key, or (2) every data value in a sample from column b has a matching value in column a. The sample used to check the condition in (2) need not be large; in our implementation the sample size was set at a few hundred rows.

The number of potential candidates is typically quite large for a complex schema with a large number of columns. Therefore CORDS applies a flexible set of heuristic pruning rules to reduce the search space and avoid discovery of spurious relationships. Some useful types of pruning rules include (1) constraints on the data type of a_1 and a_2, (2) constraints on the minimum number of distinct values of a_1 and a_2, (3) constraints on permissible pairing rules, e.g., only allow trivial pairing rules or pairing rules that correspond to explicitly declared primary-key-to-foreign-key relationships, (4) workload-based constraints, e.g., prune candidate columns that do not appear at least once in an equality predicate in a query workload.

3. Discovering Relationships

Given a candidate (a_1, a_2, P), CORDS first checks for "trivial" cases in which one or both of the columns a_1 and a_2 is "almost" a key or "almost" single valued. If the situation is nontrivial, then CORDS looks at a sample of n column-value pairs. CORDS identifies a soft FD $C_1 \Rightarrow C_2$ if $|C_1, C_2|_S \leq \epsilon_2 n$ and $|C_1|_S \geq (1 - \epsilon_3)|C_1, C_2|_S$. Here $|C_1|_S$ denotes the number of distinct values in column C_1 in the sample; $|C_1, C_2|_S$ is defined similarly, but for the concatenation of columns C_1 and C_2. The small constants ϵ_2 and ϵ_3 are parameters of the algorithm. Intuitively, CORDS asserts the existence of a soft FD $C_1 \Rightarrow C_2$ if $|C_1|_S/|C_1, C_2|_S$ is "close" to 1. Because determining FDs from samples is somewhat risky, CORDS only tests for a soft FD if the reduced table S contains enough "information" in the sense that $n \gg |C_1, C_2|_S$. Intuitively, if the sample S is so small that most column-value pairs (x, y) in the sample are distinct, then a spurious FD will likely be detected. If two columns C_1 and C_2 do not appear to satisfy a soft FD, then CORDS performs a robust chi-squared procedure to test for correlation. When constructing the two-way contingency table used for the test, the values in each column C_i are bucketized into categories. If the data distribution is highly skewed in that some values are much more frequent than others, then CORDS uses each frequent value as a category; otherwise values are assigned to categories in a uniform manner. Figure 1 displays relationships that were discovered in a database of automobile accident data; the graph was generated automatically.

4. Sample Size Determination

CORDS uses a sample size n such that, when the "mean-square contingency" measure of correlation (see [1]) exceeds a specified constant δ, the chi-squared test will correctly reject the independence hypothesis with probability at least $1 - p$. Denote by d_1 and d_2 the number of categories for the two columns of interest, and set $\nu = (d_1 - 1)(d_2 - 1)$ and $d = \min(d_1, d_2)$. Then it can be shown that, when p is small and ν is large, the required sample size is given by

$$n \approx \frac{\left[-16\,\nu \log\left(p\sqrt{2\pi}\right)\right]^{1/2} - 8\log\left(p\sqrt{2\pi}\right)}{1.69\,\delta(d-1)\nu^{-0.071}}.$$

Observe that the required sample size is essentially insensitive to the database size. Because of this insensitivity, CORDS scales well to very large databases.

5. Exploiting Relationships

CORDS exploits discovered correlations by recommending sets of CG statistics to maintain. CORDS automatically ranks the discovered soft FDs and correlations by an appropriate measure of strength (e.g., the quantity $|C_1|_S/|C_1, C_2|_S$ for soft FDs), so that only the most important statistics are stored. Figure 2 illustrates the speedup obtained by applying CORDS to a collection of test queries. As can be seen, there is no significant degradation, and some processing times are reduced by orders of magnitude.

References

[1] I. Ilyas, V. Markl, P. Haas, P. Brown, and A. Aboulnaga. CORDS: Automatic discovery of correlations and soft functional dependencies. In *Proc. 2004 ACM SIGMOD*. In press.

[2] M. Stillger, G. M. Lohman, V. Markl, and M. Kandil. LEO — DB2's LEarning Optimizer. In *Proc. 27th VLDB*, pages 19–28. Morgan Kaufmann, 2001.