REVIEW

Discovering and Exploiting Statistical Properties for Query Optimization in Relational Databases: A Survey

Peter J. Haas[1]*, Ihab F. Ilyas[2], Guy M. Lohman[1] and Volker Markl[3]

[1] *IBM Almaden Research Center, San Jose, CA, USA*

[2] *University of Waterloo, Waterloo, Ontario, Canada*

[3] *TU Berlin, Berlin, Germany*

Received 19 February 2008; revised 28 July 2008; accepted 05 August 2008
DOI:10.1002/sam.10016
Published online 16 January 2009 in Wiley InterScience (www.interscience.wiley.com).

Abstract: Discovering and exploiting statistical features in relational datasets is key to query optimization in a relational database management system (RDBMS), and is also needed for database design, cleaning, and integration. This paper surveys a variety of methods for automatically discovering important statistical features such as correlations, functional dependencies, keys, and algebraic constraints. We discuss proactive approaches in which the data is scanned or sampled (periodically, at optimization time or at query time), or in which exploratory queries are executed. Also discussed are reactive approaches that monitor the results of the query processing. Finally, we discuss methods for dealing with the practical challenges of maintaining statistical information in the face of heavy system utilization, and of dealing with inconsistencies that arise from incomplete cardinality models, use of multiple discovery methods, or changes in the underlying data over time. © 2009 Wiley Periodicals, Inc. Statistical Analysis and Data Mining 1: 223–250, 2009

Keywords: relational database; query optimization; data mining; query feedback; sampling; statistical structure

1. INTRODUCTION

One of the great breakthroughs in information management in the last 35 years was the invention and development of relational databases [1,2], which store data in tables, and a high-level, declarative language for accessing and manipulating those tables. This language, which was standardized as structured query language (SQL), permits the user to describe a desired set of data to be retrieved, without requiring the user to specify the details of how the system is to access the data. Relational databases have had, and continue to have, a major technological and economic impact; for example, the Gartner Group (Press release, June 18, 2007; see www.gartner.com) has estimated the 2006 worldwide revenue from relational-database software at $15.2 billion.

In a relational system, the rows of a table correspond to data items, such as employees, and the columns correspond to attributes of a data item, such as `AGE`, `SALARY`, and so forth. An SQL query describes a desired output table, which is computed from the set of *base tables* that are stored on disk.The computations involve execution of relational operations such as applying a selection predicate (e.g., '`DEPT.MGR = "Jane Smith"`') to the rows of a table, projecting out specified columns of a table (possibly removing duplicates in the process), and joining multiple tables together as specified by one or more *join predicates* such as '`EMP.DEPTNO = DEPT.DEPTNO`'. The latter join operation might be used, say, to create a table of employees and their managers. (Throughout, we use standard relational notation in which *R.A* denotes attribute *A* in table *R*.) Many queries also involve aggregation, i.e. computing sums, averages, and so forth over the rows of a derived or base table; the output of the query is a (possibly one-row) table of such aggregates. An aggregation query with a GROUP BY clause first groups rows together according to one or more attribute values—e.g. grouping sales transactions by state and year—and computes an aggregate separately for each group—e.g. sum of sales by state and year.

For a given SQL query, the system automatically determines a *query execution plan* (QEP) for actually retrieving the data. A QEP can be conveniently represented as an upside-down tree, with the leaves corresponding to the base

Correspondence to: Peter J. Haas (peterh@almaden.ibm.com)

tables and the root corresponding to the final query results. Because relational operators generally can be applied in many different orders, and because each operator can be executed in different ways, the number of possible QEPs for a given query can easily number in the hundreds or even thousands, each producing identical results. The *query optimizer* [3–7] is the system component that selects the cheapest QEP, using a sophisticated performance model to estimate the execution cost of alternative QEPs.

This paper surveys various statistical and data-mining methods for supporting query optimization in a relational database management system (RDBMS). We focus on techniques for automatically mining statistical information from the data, and for using this information either directly in query optimization, or indirectly, to automatically select and maintain the statistics used in the query optimizer's 'cardinality model.'

1.1. Query Optimization

Figure 1 shows two alternative QEPs for a query in which tables R, S, and T are joined together via a sequence of dyadic join operators, with a local selection predicate on T, namely, $T.A > 3$. The two execution plans differ in several ways:

- QEP(a) joins tables R and S first then joins the output with table T, whereas QEP(b) joins R and T first and then joins the output with S.
- QEP(a) accesses the rows of S by sequentially scanning the entire table, whereas QEP(b) uses an *index* structure to efficiently access specific rows of the table, as required by the subsequent join operation.
- QEP(a) applies the selection condition $T.A > 3$ as a filtering operation following the sequential scan of T, whereas QEP(b) uses an index on attribute $T.A$ to access only qualifying rows.

Figure 2 illustrates a typical index structure, called a B^+ *-tree*, which supports efficient retrieval of rows based on the indexed attribute(s)—`Name`, in the illustrated example. Rows having a specified attribute value are located by traversing the tree from root to leaf and then to the file containing the actual data, where the pointer to follow at each internal node is determined by comparing the target attribute value to the values stored at the node. (The leaf nodes have additional 'horizontal' pointers to support efficient retrieval of rows that correspond to ranges of consecutive attribute values). Sometimes, as in the example, the index nodes store other information beneficial to a query optimizer, such as the number of rows having a given key value (e.g. 'Costa'). Typically, the leaf nodes are stored on

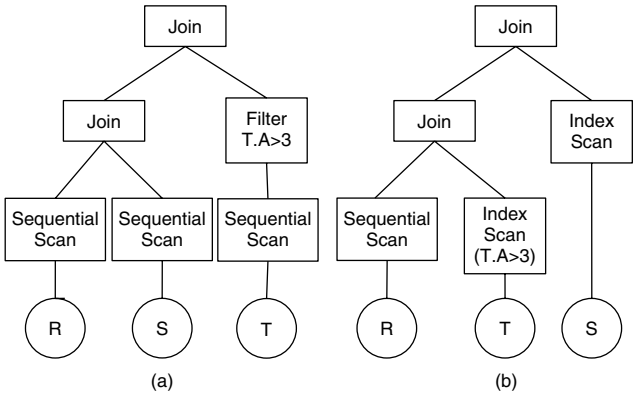

Fig. 1 Two QEPs.

disk pages and the internal nodes are cached in memory (in the *buffer pool*). So one or two additional I/Os may be required to retrieve a particular record using an index, but index-based retrieval avoids the need to access all of the pages in the table via a sequential scan. The relative efficiency of the above two QEPs depends mostly upon the number of rows in the table that satisfy the selection predicates on that table. Generally, I/O costs dominate CPU costs in database query processing. Accessing the rows in a table using an index, if available, is the most I/O-efficient option if less than about 10% of the rows must be retrieved; otherwise, use of a simple table scan yields lower I/O costs. Each dyadic join operation can potentially be executed in a variety of ways, for example, sorting the rows in each table by the join attribute and then merging, or hashing on the join attribute in each table, or simply executing a nested loop where each row in the 'outer' table initiates a sequential scan of the rows in the 'inner' table to find matches. Again, the cost of the algorithms depends strongly on the number of rows processed by each operation. For example, if we use a nested-loop algorithm for the topmost join in QEP(a) and assume that each of the two (nonbase) input tables have previously been written to disk, then the cost of the join is roughly $c(|O| + |O||I|)$, where $|O|$ and $|I|$ are the size of the outer and inner tables (in units of disk pages), and c is the cost of reading a disk page. Clearly, the smaller of the two tables should be chosen as the outer, but the size of the tables depends on how many rows in T satisfy the predicate and how many rows are produced by the join of R and S, which in turn depends on the distribution of join-attribute values in each table.

A query optimizer selects the cheapest QEP based on a sophisticated performance model that estimates the execution cost of alternative QEPs. Most commercial query optimizers generate many legal QEPs, model the expected execution cost of each, and choose the cheapest. QEP generation, costing, and comparison are typically performed in an

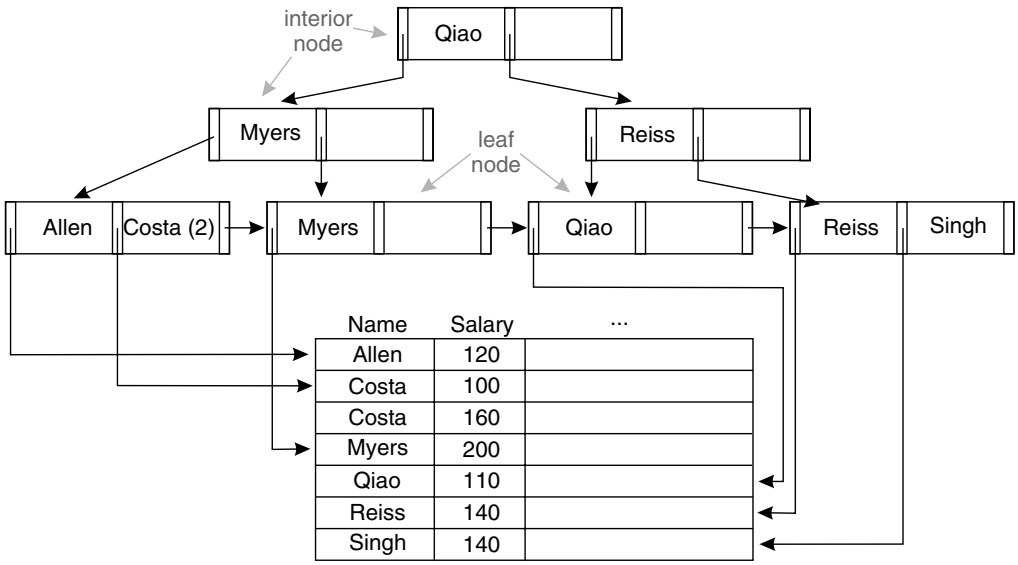

Fig. 2 A B^+-index on the `Name` attribute.

incremental manner, from leaves to root, often in the context of a dynamic programming or greedy search algorithm. Important inputs to the performance model are statistics about each database table (e.g. number of rows), column (e.g. maximum, minimum, and number of distinct values), and index (e.g. number of levels and number of leaf pages), which are typically collected periodically by a batch utility during periods of low system utilization. As noted in the example above, the cost of each QEP largely depends upon the number of rows to be processed; whether the QEP will access three rows or 3 000 000 rows matters more to the optimizer's decision than the precise cost per row. The performance model can therefore be divided into a *cardinality model* that estimates the number of rows to be produced by each operator in a QEP, and a *cost model* that estimates the execution cost of an operator, given the estimated number of rows. (The latter model takes as input factors such as data layout on disk, disk-access speeds, and so forth). Modern cost models have been validated to be quite accurate when the cardinalities are known, but as we shall see in the next section, cardinality models are laced with simplifying assumptions that can introduce order-of-magnitude errors. Hence much of the literature on query optimizers has concentrated on improving cardinality models, and this paper will do likewise. However, it should be noted that accurate cardinality estimates are necessary, but not sufficient, for the query optimizer to choose the best QEP.

The effectiveness of a query optimizer is usually measured using a benchmark, in which a dataset is loaded into relational tables and a workload of queries is issued to the query engine. There is little consensus within the database community on benchmarks, due to both the wide range of applications in which RDBMSs are used and the difficulty of finding a collection of realistic-enough complex queries over a given dataset. (Indeed, in some industries, query workloads are considered trade secrets). The only truly standard benchmarks for query optimizers at this time are the TPC-H and TPC-DS benchmarks for business intelligence and decision-support queries administered by the Transaction Processing Performance Council (www.tpc.org). The datasets for these benchmarks are synthetically generated. Even for these relatively simple benchmarks, determining the truly optimal plan for each query has proved impossible, so that competing optimizers can be compared against each other, but not against ground truth.

1.2. Cardinality Models

As indicated above, an optimizer's cost model relies heavily upon the underlying cardinality model. The cardinality model is used to estimate the amount of data that will 'survive' each predicate in a QEP, e.g. the fraction of rows in a table that will satisfy a given selection predicate on attributes in the table. This fraction is called the *selectivity* of the predicate, and can also be interpreted as the probability that a randomly selected row will satisfy the predicate. The selectivity is multiplied by the *table cardinality* (i.e. the number of rows in the table) to estimate the total number of rows that will be subject to further processing after the predicate is applied. Similarly, the selectivity of a join predicate between two tables is defined as the fraction of row-pairs in the Cartesian product that participate in the join. For example, the Cartesian product of the two tables in Fig. 3 comprises $5 \times 2 = 10$ row-pairs, whereas only 5 row-pairs satisfy the join predicate `Employee.Deptno = Manager.DeptNo`, so that this predicate has a selectivity of

Employee

Name	DeptNo
Rao	100
Smith	100
Abbas	100
Markl	200
Reiss	200

Manager

Name	DeptNo
Iyer	100
Jones	200

Fig. 3 Join-selectivity example.

50%. (Here `Manager.DeptNo` is a *key*, in that there are no duplicate values, and `Employee.DeptNo` is a *foreign key*, in that each `Employee.DeptNo` value appears in the `Manager.DeptNo` key column). The cardinality model can be viewed as approximating the joint frequency distribution of all of the attributes in the database. Such an approximate model is needed because databases can have hundreds of tables, each having hundreds or even thousands of columns, and so it is impractical to store the entire joint distribution.

Early RDBMSs used a very crude cardinality model that could handle any query, but not terribly well [7]. These systems maintained a minimal 'thin veneer' of statistics in the system catalog. For example, for each table, the system recorded the table cardinality and, for each column, the 'column cardinality,' that is, the number of distinct values. The cardinality model then assumed that the data distribution was uniform within each column—i.e. that the distinct values were equi-frequent—and that columns were mutually independent. Thus, if a table had 400 000 rows and the columns `MAKE` and `MODEL` had respective column cardinalities of 10 and 400, the selectivity of the predicate '`MAKE = Honda AND MODEL = Accord`' would be estimated as $(1/10)(1/400) = 1/4000$, so that approximately $(400000)(1/4000) = 100$ rows would be expected to satisfy this predicate. Note that such an estimate is likely to be very poor, because MAKE and MODEL are related by essentially a *functional dependency* (FD), in that MODEL determines MAKE (outside of a few exceptional cases). Thus, even if the distinct MODEL values were truly equi-frequent (which is unlikely), the true selectivity would probably be closer to $1/400$, so that the naive estimate would underestimate the true selectivity by an order of magnitude and the processing cost would be much higher than expected.

To handle an 'equijoin' between two tables R and S, that is, a join based on an equality predicate of the form '$R_1.A = R_2.B$,' the crude cardinality model assumed that there was an *inclusion dependency* between the join columns, i.e. that every distinct value in the smaller join column was assumed to appear in the larger join column. Here 'smaller' and 'larger' refer to column cardinalities. Thus, if $d_1 < d_2$, where d_i is the column cardinality of table R_i, then each distinct value in R_1 occurs $|R_1|/d_1$ times in R_1 and $|R_2|/d_2$ times in R_2, for a total contribution of $|R_1 \times R_2|/(d_1 d_2)$ rows to the join. (We have used the uniform-distribution assumption here). Allowing for the fact that there are d_1 such values, and considering the general case in which d_2 may be smaller than d_1, we obtain the general selectivity estimate $\sigma = \min(d_1, d_2)/(d_1 d_2)$; this estimate generalizes directly to joins involving three or more tables.

As indicated by the foregoing automobile example, the classical uniformity and independence assumptions can yield atrociously bad selectivity estimates in the presence of 'skewed' data—i.e. where the distinct values have widely varying frequencies—or correlations between columns. (Here 'correlations' refers to general statistical dependencies, not necessarily linear relationships as measured, for example, by the Pearson correlation. We will use this somewhat imprecise, but common, database terminology throughout). Such bad estimates, in turn, can cause the optimizer to choose a highly suboptimal QEP; in real applications, queries that should take seconds or minutes to complete can take hours or days. Consequently, as database systems have matured, RDBMS designers have made better efforts to approximate the true joint frequency distribution of the attributes.

To avoid the uniformity assumption, designers have added more detailed distribution statistics for each column. For numerical data, the distributional information is typically represented by explicit histograms, either in the form of buckets and counts or in the form of quantiles. For discrete numerical data and categorical data, the system records the frequencies of the k most frequent data values, where k would typically be a number on the order of 10. Frequent values are used for better estimating the selectivity of equality predicates such as '`MAKE = Honda`', and the histogram gives estimates of range predicates such as '`AGE BETWEEN 25 AND 50`'.

Correlations between columns have traditionally received much less attention, due to the inherent difficulty of capturing joint distributions in the face of restrictive CPU and memory constraints. This situation has been changing, as RDBMS designers have realized that, in practice, the most serious cardinality estimation errors are often caused by erroneously assuming independence. As indicated by some of the experimental results that are cited in the sequel, processing times for expensive queries can be reduced by orders of magnitude by even roughly capturing correlation information. One early solution to dealing with correlation was to partition columns into correlated groups and maintain a set of 'column-group' (CG) statistics for each group. For a given column group, the CG statistics coincide with the usual single-column statistics, but are computed on the concatenation of the columns in the group. Whenever detailed information was lacking, the uniformity and independence assumptions would be imposed to fill in the gaps.

More recently, commercial systems have tried to exploit distributional information not only on base tables, but also on specified derived tables corresponding to partial query results that appear frequently in the workload; the resulting set of statistics are sometimes called *statistical views*, (In database terminology, a *view* over the base tables is a derived table that is specified by an SQL query. *Materialized views* are instantiated and stored on disk, usually with the goal of speeding up query processing. Statistical views have statistics, but not actual data, associated with them) and partially capture joint-distribution information.

1.3. Automated Statistics Configuration

Traditionally, it was left to the user—or to a highly skilled database administrator (DBA)—to determine which optimizer statistics to maintain. This type of labor-intensive activity has been a major component of the total cost of ownership for an RDBMS. Database designers have therefore expended major efforts over the past decade to develop methods for *automated statistics configuration*, that is, methods for the system to automatically decide which statistics to maintain, and when to collect and refresh these statistics. The decisions that need to be made include: the attribute sets on which to maintain joint statistics; the numbers of frequent values, histogram-bucket frequencies, or quantiles to collect; the choice of bucket boundaries for histograms; and the set of statistical views to materialize. There have also been efforts to make the statistics-collection process more dynamic, gathering information during query execution so that the QEP can be modified on the fly.

The key step in determining how to configure optimizer statistics is to understand the statistical structure of the data. The primary high-level features of this structure, discussed in the sequel, include identification of 'important' correlations between attributes, significant departures from the uniform distribution, and the presence of (possibly 'soft') keys, functional dependencies, inclusion dependencies, and algebraic constraints. Here 'importance' and 'significance' can be with respect to a given user workload, or with respect to all possible future workloads. Besides improving selectivity estimates, structural features can also be used to provide new access paths to the optimizer by permitting queries to be rewritten to new queries that have the same answer, but can potentially be computed more efficiently. In the following sections, we therefore describe techniques not only for selecting and gathering optimizer statistics, but also for automatically discovering useful high-level statistical features, both for direct use in the optimizer and for automated statistics configuration.

1.4. Proactive and Reactive Approaches

Traditionally, users have been responsible for initiating the collection of statistics proactively by invoking a utility that scans the tables and indexes in the database. But even this simple approach has variants. Should the statistics to be collected favor those needed for a given query workload, either because the same workload is routinely repeated, or because an historical workload is often indicative of workloads anticipated in the future? Need we obtain exact statistics by scanning every table and index in its entirety, or would sampling of the data acquire statistics that are 'good enough' for the purposes of query optimization?

A more recent, complementary approach is to opportunistically collect statistics during query execution. This reactive approach has the benefit that better information is obtained at lower cost on portions of the database that are accessed frequently, but leaves the optimizer less informed for portions of the database touched by queries that were not anticipated by previous queries.

Proactive and reactive approaches can be applied not just to collecting basic statistics, but also to discovering higher-level statistical features such as correlations and functional dependencies. We discuss these two major approaches below. Section 2 presents some proactive approaches to statistics collection and feature discovery, both workload-aware and workload-blind. We then detail several reactive methods in Section 3.

1.5. Exploitation and Maintenance

In the third major section of the paper (Section 4), we discuss some practical challenges that arise when trying to exploit and maintain the discovered statistical information.

A key problem is that multiple forms of discovery may be used simultaneously, e.g. data scanning together with query feedback, leading to a variety of partial distribution information. Use of this information raises the possibility that multiple, nonequivalent selectivity estimates may be available for a given predicate. Traditional optimizers have used cumbersome ad hoc methods to ensure that selectivities are estimated in a consistent manner. Besides being expensive and inefficient, these methods ignore valuable information and tend to bias the optimizer toward QEPs for which the least information is available, often yielding poor results. A related problem is that the available pieces of partial distribution information may not even be consistent with each other. We describe some recently proposed methods that use maximum entropy ideas and linear programming techniques to address these issues.

Finally, an important challenge is how to maintain statistical information over time, taking into account the load on the system, user response requirements, the degradation of

information over time, and the relative importance of the various statistics. We describe one initial attempt to deal with these issues in a commercial RDBMS, namely, IBM's DB2 database system for Linux, Unix, and Windows (called 'DB2 for LUW' in the sequel). The paper concludes with some ongoing challenges and directions for future research.

1.6. The Scope of Our Discussion

Although we try to at least touch on most of the techniques that have been studied by the database community, we focus most strongly on techniques that have been implemented, or at least seriously prototyped, in commercial systems, where data volumes are approaching petabytes and response-time requirements are becoming increasingly stringent. This setting imposes some tight constraints on the discovery methods that can be used. Most modern systems run continuously, with no down time, so that discovery algorithms often must be run in background mode, and must therefore not consume too many system resources; based on our experience with DB2, a query-processing-time overhead of at most 5% seems to be acceptable in practice. Moreover, to minimize the cost of code development and maintenance, the algorithms must not be overly complex. A major challenge for commercial systems is thus to find—along the continuum between simplistic assumptions of independence and uniformity and elaborate algorithms for maintaining a precise approximation to the complete joint attribute–value distribution—the ideal 'sweet spot' that optimally balances simplicity, accuracy, and speed. We discuss reactive techniques in somewhat more depth than proactive techniques, because (i) there are fewer such techniques to cover, (ii) such techniques are relatively more recent, and (iii) we feel that they are more representative of where current database technology is heading.

Our practical orientation also leads to a focus on the sorts of histogram-type statistics used in current and impending systems. Researchers have pursued many other sophisticated types of database synopses, including wavelets [8], sketches [9,10], neural nets [11,12], and probabilistic relational models [13], but evaluating the advantages and disadvantages of all of these is beyond the scope of this paper. We do, however, try to make our survey appealing to readers oriented toward statistics and data mining, and so tend to devote space to the various techniques in proportion to their expected statistical interest rather than to the impact that they have had so far on the commercial database industry. In particular, we highlight maximum-entropy techniques, because these methods seem to strike a nice balance between statistical sophistication and practical feasibility in industrial-strength systems.

Commercial database systems are large, complicated, and used in a wide variety of applications. We therefore need to cover a wide variety of somewhat specialized topics. Figure 4 may help the reader keep track of how the many techniques that we discuss fit together in the context of an RDBMS. The items in gray correspond to the topics that are central to this survey, because they are either crucial to the optimizer, the focus of current research, or of potential interest to the statistics and data mining communities. We emphasize that no existing RDBMS incorporates all of these techniques, but most of them have at least been prototyped in real systems.

The problems and solutions of this paper are couched in the terminology of relational databases, because most of the original work was performed in this context. However, many of the techniques are applicable to systems in which data items are stored as complex 'objects' in the form of hierarchies. Examples include object-oriented, object-relational, and XML DBMSs. A key observation is that object hierarchies can be viewed as joined, or *de-normalized*, tables. For instance, using our earlier car example, an object-based system might store a `CARS` object made up of several `MAKE` objects, each containing numerous `MODEL` objects. Such hierarchies make clearer the correlations between models and makes, but the problem of estimating the number of objects that will satisfy a query against the `CARS` object is roughly equivalent to the problem of estimating selectivities of predicates on the table formed by joining the original `MAKE` and `MODEL` relational tables. See Balmin et al. [14] for a discussion of query-optimization issues in the XML setting.

We note that, although the primary focus of this paper is the discovery and exploitation of statistical features by

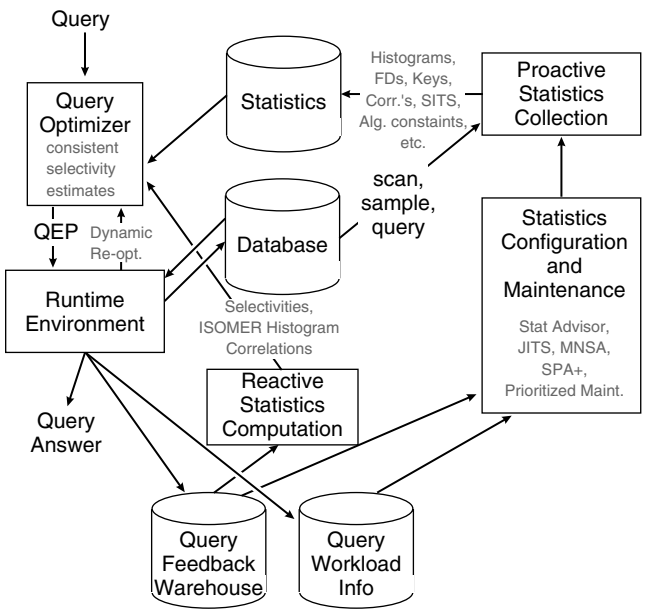

Fig. 4 Components of an RDBMS.

the SQL query optimizer, the discovered correlations, keys, and other statistical relationships may also facilitate other database functions, such as cleansing the database, mining it for trends or anomalous conditions, designing optimal data layouts on disk, designing optimal physical structures for accessing the data, or integrating multiple, heterogeneous databases into a coherent whole. Another important application is query optimization in parallel databases, where knowledge of the frequency distribution of a column can be used to evenly distribute the rows of a table among processors when using hash partitioning; similarly, identification of keys can be useful for data partitioning. Finally, we acknowledge an inevitable bias towards work performed by the authors or their immediate colleagues, since we know this material best.

2. PROACTIVE APPROACHES

Proactive approaches obtain statistical information by actively scanning or sampling the data, or by issuing exploratory queries. Traditionally, this data exploration is performed at periodic intervals—see Section 4.2—although more recent approaches, described in Section 2.2 below, may collect data whenever a query is optimized. When determining the data to examine or the statistics to collect, proactive methods can either try to exploit information about the query workload, or can ignore the workload. Workload-aware methods differ from reactive methods in that they exploit knowledge of the queries that are to be executed, but not knowledge of the actual answers to the queries. As with proactive versus reactive approaches, however, workload-aware methods provide more accurate information than workload-blind methods when the true workload matches the workload assumed by the discovery process. Workload-blind methods, while often less accurate, are more robust to time-varying changes in, or unexpected deviations from, the assumed workload. We give examples of these various approaches below.

2.1. Workload-blind Methods

Workload-blind proactive methods can be further categorized by the methods used to access the data. The simplest, though most expensive, approach is to completely scan all of the data. As database sizes have increased, sampling-based methods, when applicable, have become increasingly popular because of their desirable scalability properties. An alternative approach is to proactively execute training queries, in order to learn a high-level cardinality model that encapsulates important statistical structure.

2.1.1. Scan-based methods

Classical statistics As discussed in Section 1, query optimizers have from their origins collected a set of basic statistics on all tables in the database, and on all columns and indexes pertaining to each table. These basic statistics are used in both the cardinality and cost models of the optimizer, and are collected via a utility (called RUNSTATS in IBM's DB2 family of products) that scans each table and each index on that table once per execution.

Besides the distributional statistics mentioned in Section 1 (frequent values, 1D-histograms, quantiles), RUNSTATS collects the number of pages in the table, in order to estimate the I/O cost to access the table by a simple table scan. Moreover, for each numerical column, RUNSTATS collects the second highest and lowest values and the average width of variable-width columns, as well as the column cardinality. The second highest and lowest values are used to estimate the selectivity of numerical range predicates as a portion of a uniform distribution between these two extreme values, assuming implicitly that the highest and lowest values are outliers.

Scan-based, exact computation of frequent values and quantiles is a daunting task, because straightforward approaches—e.g. that compute quantiles via sorting the table—are too expensive with respect to CPU and/or memory to be practical. DB2 for LUW therefore computes approximations to these statistics by collecting a reservoir sample [15] during the table scan, and then estimating frequent values and quantiles by sorting the sample on each column of interest. If the presence of frequent values causes multiple quantiles to coincide, the frequent values are separated out into individual histogram buckets to create a 'compressed' histogram—see Poosala et al. [16] for details. Also see Greenwald and Khanna [17] and Zhang and Wang [18] for approximate scan-based approaches to quantile estimation; these latter approaches have not yet been incorporated into commercial systems.

The general problem of constructing multidimensional histograms for approximating the joint distribution of data values has received much attention from the database research community; see Ioannidis [19] for a survey. This problem is extremely challenging, and most proposed methods are too expensive to be practical, requiring multiple passes over the data. Even when avoiding scans by using sampling—see, e.g. Muralikrishna and DeWitt [20]—the problem of defining the histogram buckets is nontrivial. Several promising approaches have emerged for proactive, one-pass methods for constructing multidimensional histograms [21–23], but none have been fully investigated in the context of commercial systems. See also the discussion of graphical statistical models below, where, instead of maintaining a single multidimensional histogram,

the systems maintain a carefully selected set of low-dimensional histograms.

As with frequent values and quantiles, exact computation of the column cardinality via sorting or hashing is impractical. Most systems compute an estimate of column cardinality based on a single scan of the table, using limited memory; see Beyer et al. [24] for a recent review of these techniques. An important problem that arises in parallel database systems, when the rows of a table have been distributed among a set of processing nodes, is how to take a set of statistics that have been computed at these nodes and combine them into an overall statistic for the entire table. The work in [24] addresses this problem in the context of column-cardinality estimation.

Many techniques for collecting statistics on tables can be adapted to statistics collection on indexes. As indicated previously, index statistics include the number of leaf pages, i.e. pages of leaf nodes, the number of distinct values of the index's key column(s), and the number of levels in the B^+ tree that implements the index. These are used in the cost model to estimate the number of I/Os to access the table via an index, either when a key range is specified by a predicate in the query and/or when the index is used to provide the rows in order for an ORDER BY clause, a GROUP BY clause, or for purposes of a sort-merge join.

Functional and other dependencies. Recall from the `MAKE` and `MODEL` example of Section 1.2 that knowledge of a functional dependency—i.e. knowledge that the value of an attribute determines the value of another attribute—permits much more accurate selectivity estimates. Inclusion dependencies are similarly valuable for selectivity estimation; recall from Section 1.2 that such a dependency exists between columns C_1 and C_2 if every distinct value in C_1 appears in C_2. A number of researchers have therefore developed scan-based methods for detecting such relationships. For example, the TANE algorithm [25] uses all of the data to search for generalized FDs of the form $\{a_1, a_2, \ldots, a_n\} \rightarrow b$, where this notation means that $\{a_1, a_2, \ldots, a_n\}$ is the minimal set of attributes whose values uniquely determine the value of attribute b. The algorithm uses a level-wise approach reminiscent of the Apriori algorithm for association-rule mining [26]. The algorithm also finds FDs that are approximate, in that they hold after a small number of rows are removed from the table. As another example, the algorithms in Bell and Brockhausen [27] and in Petit et al. [28] execute a sequence of queries involving joins and `COUNT(DISTINCT)` operations to discover inclusion dependencies.

Algebraic constraints. Another important type of statistical structure is the presence of an *algebraic constraint*. Such a constraint describes a strong correlation between database attributes, and hence can be used to avoid the independence assumption when estimating selectivities. Perhaps more importantly, algebraic constraints can sometimes be exploited to rewrite a query into a logically equivalent form that can be processed more efficiently. For example, consider a table `Orders` with attributes `OrderDate` and `ShipDate`, and suppose that we discover the following algebraic relationship:

$$\texttt{ShipDate} - \texttt{OrderDate} \in [1, 5].$$

That is, an item is shipped between 1 and 5 days after the initial order is placed. Given a query containing the predicate '`OrderDate = 3-12-2007 AND ShipDate = 8-12-2007`,' the optimizer would know that it should avoid multiplying the selectivity estimates of the two conjuncts by assuming independence, and instead maintain joint statistics on these attributes. The foregoing algebraic constraint can also be used for query rewriting. For example, consider a query that selects all orders with '`OrderDate = 1-12-2007`.' If an index is available on `ShipDate` but not on `OrderDate`, the query can be rewritten with a new, equivalent predicate '`ShipDate BETWEEN 1-13-2007 AND 1-17-2007`,' to make use of the index. See Godfrey et al. [29] for further discussion of how to exploit constraints in an RDBMS. Gryz et al. [30] discover constraints of the form $bX + a_0 < Y < bX + a_1$ between attributes X and Y in a table by performing a linear least-squares fit to the (X, Y) value pairs, calculating the maximum absolute deviation of the values in the table from the fitted line, and retaining the constraint if this deviation is less than a specified upper bound.

Holes in joins. Gryz and others [31,32] have developed a scan-based algorithm for discovering 'holes in joins.' A hole in a join between tables R and S is defined in terms of a range $[x_0, x_1]$ of values for an attribute $X \in R$ and a similar range $[y_0, y_1]$ for an attribute $Y \in S$ such that no elements in the Cartesian product of R and S satisfy the predicate '$R.X$ `BETWEEN` x_0 `AND` x_1 `AND` $S.Y$ `BETWEEN` y_0 `AND` y_1.' Similarly to algebraic constraints, knowledge of such empty regions can be used both to obtain better selectivity estimates—because the presence of holes indicates correlations between attributes—and to rewrite queries for greater processing efficiency. For an example of the latter technique, consider a query of the form

SELECT $*$ FROM R, S
WHERE $R.X$ BETWEEN u_0 AND u_1
AND $S.Y$ BETWEEN v_0 AND v_1.

Suppose that a hole of the above form is known and that $x_0 < u_0 < x_1 < u_1$ and $y_0 < v_0 < v_1 < y_1$ (Fig. 5). Then the query can be rewritten as

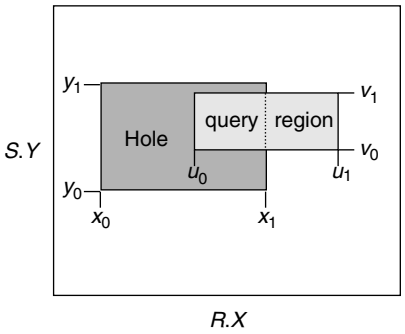

Fig. 5 A hole in a join.

SELECT * FROM R, S
WHERE $R.X$ BETWEEN x_1 AND u_1
AND $S.Y$ BETWEEN v_0 AND v_1,

without changing the answer to the query. The latter query is potentially more efficient to execute, because the predicate on $R.X$ is more selective, and hence the number of disk accesses to R can be reduced. Experiments using real-world workloads [32] have demonstrated query-processing speedups of up to a factor of four, and reductions in selectivity-estimation errors that exceeded five orders of magnitude.

Semantic integrity constraints. Holes in joins are an example of a class of constraints that involve specific attribute values, rather than being constraints on the attributes as a whole. Such constraints are sometimes called *semantic integrity constraints*, and are used for 'semantic query optimization.' For example, Gryz et al. [30], besides giving algorithms for discovering algebraic constraints, also give a scan-based algorithm for discovering semantic constraints of the form $X = a \rightarrow Y \in [y_0, y_1]$. (Here $\rightarrow$ denotes logical implication). As another example, Siegel et al. [33] and Yu and Sun [34] consider methods for discovering constraints of the form $A \rightarrow B$ and $JC \rightarrow (A \rightarrow B)$, where JC is a join condition (i.e. predicate) and $A \rightarrow B$ is a rule such as '`s.city = chicago` $\rightarrow$ `t.weight > 200`'. These latter approaches use machine learning techniques, and have not yet been applied in a commercial RDBMS. We also note that association rules as in Srikant and Agrawal [35] are closely related to semantic integrity constraints, but this topic is beyond our scope.

Keys. Another important statistical characteristic is the set of keys in a table. In general, a key is a set of columns whose concatenated values contain no duplicate rows, and serve to uniquely identify a data item. The knowledge of keys can be used to (i) provide better selectivity estimates, (ii) provide a query optimizer with improved access paths that can lead to speedups in query processing, e.g. by avoiding unnecessary sorts, (iii) allow the DBA to improve the efficiency of data access via physical design techniques such as data partitioning or the creation of indexes and materialized views, (iv) provide new insights into application data, and (v) automate the data-integration process [36]. Traditionally, all relevant keys were assumed to be known to the DBA and explicitly declared to the system. In real-world scenarios with large, complex databases, an explicit list of keys is often incomplete, if available at all. Many keys or approximate keys, although potentially valuable, are unknown to the RDBMS because

- the key represents a 'constraint' or 'dependency' that is inherent to the data domain but unknown to both the application developer and the DBA;
- the key arises fortuitously from the statistical properties of the data, and hence is unknown to the application developer and DBA;
- the key is known and exploited by the application without the DBA explicitly knowing about it;
- the DBA knows about the key but for reasons of cost chooses not to explicitly identify or enforce it; or
- the key is approximate in that the number of duplicates, though nonzero, is very small.

Many of the unknown keys are *composite* keys, that is, keys consisting of two or more attributes. The unknown keys in a database represent a loss of valuable information.

The GORDIAN algorithm in Sismanis et al. [37] efficiently computes all keys in a table. The basic idea behind GORDIAN is to formulate the problem as a datacube[1] computation problem [38] and then to interleave the cube computation with the discovery of all *nonkeys*—sets of attributes that are *not* keys. Finally, GORDIAN efficiently computes the complement of this collection, yielding the desired set of keys. The rationale for this approach is that nonkeys are easier to discover than keys: a putative key, based on the data seen so far, can be invalidated as more data is scanned (due to observation of previously unseen duplicate values), whereas the nonkey property, once discovered, will never be invalidated. Although the general problem of discovering a minimal composite key—i.e. a composite key with the fewest possible number of attributes—is NP-complete [39], GORDIAN appears to have good 'typical case' behavior on real-world datasets. For example, when restricted to a class of datasets in which attribute-value frequencies follow the generalized Zipf distribution, GORDIAN has a time

[1] A datacube represents the aggregation of a 'measure' variable along different 'dimensions.' For example for a 'sum' datacube with measure variable *sales* and with time and geography dimensions, a given 'cell' might contain the total sum of sales for a given month and for a given set of cities. Contingency tables in statistics correspond to 'count' datacubes.

complexity that is polynomial in both the number of rows and columns.

Graphical statistical models. Several authors have proposed the use of graphical statistical models to capture the global correlation structure in relational data. Such dependency information can be used for purposes of statistics configuration. Moreover, such models embody a concise representation of the joint frequency distribution of the attributes in a table, which can be used directly for selectivity estimation.

For example, Deshpande et al. [40] first capture the correlation structure of the data by means of a 'decomposable interaction model.' Such a model can be represented graphically as a *Markov network*, i.e. an undirected graph in which the nodes represent attributes and an edge represents a direct correlation between the attributes that it connects. Two attributes that are separated by an attribute A are conditionally independent, given A. The Markov network serves to define a collection of (possibly nondisjoint) groups of attributes, which correspond to the *cliques* (maximal completely connected subgraphs) of the network. A joint histogram is maintained for each clique, and desired joint distributions are computed by combining marginal frequencies for the various histograms according to rules that are determined by the interaction model. In one simple scenario, for example, the cliques correspond to disjoint groups of correlated attributes, and attributes in different cliques are considered to be independent. Thus, if a table contains attributes A, B, C, and D and there are two cliques $[A, B]$ and $[C, D]$, then the selectivity of the predicate '$A = a$ AND $D = d$' would be computed as $f_{AB}(a, +)f_{CD}(+, d)$, where f_{AB} and f_{CD} are the joint relative frequency functions encapsulated in the histograms on the two cliques, $f(a, +) = \sum_{b \in B} f_{AB}(a, b)$, and $f_{CD}(+, d) = \sum_{c \in C} f_{CD}(c, d)$. More generally, the model can deal with conditional independence relationships, e.g. in which attributes A and C are independent, given the value of a third attribute B. For this example, the cliques would be $[A, B]$ and $[B, C]$, and the selectivity of the predicate '$A = a$ AND $B = b$ AND $C = c$' can be computed as $f_{AB}(a, b) f_{BC}(b, c)/f_{BC}(b, +)$ or, equivalently, $f_{AB}(a, b) f_{BC}(b, c)/f_{AB}(+, b)$. To limit computational complexity, cliques are constrained to contain at most three or four attributes. The interaction model is fitted using a heuristic 'forward selection' search process, in which full independence is assumed initially—so that there is exactly one singleton clique per attribute—and cliques are built up incrementally based on improvements in the approximation, as measured by decreases in Kullback-Liebler distance. For a given proposed interaction model, the number of buckets per histogram is allocated using a greedy algorithm, and bucket boundaries within each histogram are also selected in a greedy fashion. The resulting overall candidate model is then scored.

Getoor et al. [13] explore an approach somewhat similar to the one above, but based on a directed *Bayesian network* representation, in which a given attribute is independent of its nondescendants, given its parents. In this framework, the relative frequency distribution is represented via a set of conditional univariate probability distributions, where the distribution of a given attribute is conditioned on the values of its parents. To handle joins, the authors extend the basic model to allow the parents of an attribute in a given table R to include attributes in another table S, provided that R has a foreign key that points to S—recall from Section 1.2 that column F in R is a foreign key that points to S if S has a key column K and every value in $R.F$ appears in $S.K$. The authors call this model a *probabilistic relational model* (PRM); see Getoor and Taskar [41] for a recent discussion of such models. As in [40], a PRM is fitted to the data using a heuristic search through possible correlation structures. For a given candidate structure, the needed conditional probabilities are simply estimated via sample frequencies when the column cardinality is not too large; otherwise, bucketizing techniques analogous to those in [40] must be used.

Although the foregoing approaches can yield very accurate selectivity estimates, graphical statistical models have not yet been incorporated into commercial systems. The key drawbacks seem to be the relatively high coding effort and level of sophistication required, as well as the relatively high computational expense of the approach both in fitting the model and in computing estimates during optimization.

Statistical learning techniques. A final class of proactive techniques represents the statistical features of the data indirectly, via a learned statistical model. This approach defines a vector **v** of 'features' that concisely characterizes the query, the dataset, and the available system resources. A set of exploratory queries is executed, corresponding to a variety of values of **v**, and the corresponding query cost **c** is observed for each **v**. The resulting set of **(v,c)** pairs is used to learn a statistical model that predicts the query cost from the input feature values. Babu et al. [42] argue that a proactive approach involving exploratory queries, rather than a reactive approach involving user queries, is needed in order to cover a broad enough range of possible query-processing conditions. Some statistical learning approaches that have been considered in the database research community include linear and nonlinear regression [43–45], interpolation schemes [46], neural nets [11,12], and, more recently, transform regression [47]. The statistical learning approach has so far been proposed primarily for estimating the cost of expensive user-defined functions, and in the context of multidatabase systems and XML systems, but

the technology could potentially be applied directly in the RDBMS setting.

2.1.2. *Sampling-based methods.*

As databases become ever larger, proactive techniques that require a full scan of the data become harder to use in practice, especially if—as is often the case—these techniques are not massively parallelized. Consequently, there has been an increasing amount of interest in sampling-based methods. For example, the RUNSTATS utility in DB2 for LUW can now be configured to sample the data, rather than execute a full scan. In general, any of the foregoing scan-based methods can in principle simply be applied to a sample of the data—e.g. sampling has been proposed for use with the GORDIAN key-finding method discussed in Section 2.1.1—but the resulting accuracy of this approach is not well understood in general.

We now describe several recent methods for discovering statistical features in which sampling techniques are integral to the algorithms. Almost all of the algorithms described below assume that a simple random sample of the rows in a table is available. Such a sample is typically too expensive to compute on the fly. For example, if a disk page contains 100 rows, a simple calculation shows that collecting a sample of 2% of the rows would require that roughly 87% of the pages be fetched, and the resulting I/O cost would be close to that of a full scan. One possibility is to amortize the cost of sampling by incrementally maintaining a random sample; Gemulla et al. [48,49] provide several algorithms for this purpose. Another possibility is to simply use page-level sampling, and adapt the sampling-based procedures to deal with the statistical correlation that can result when rows are assigned to disk pages nonrandomly, e.g. based on their column values. This approach is analogous to the 'cluster sampling' techniques found in the finite-population sampling literature; see Haas and König [50] for a discussion of page-level, row-level, and hybrid sampling schemes for relational databases. Such extensions of sampling-based algorithms from row- to page-level sampling can be quite challenging, and have received relatively little study so far. In the context of parallel database systems, it is often desirable to obtain samples of the rows of a table independently at each processing node and then combine these samples into an overall sample of the rows in the table. This issue has been explored by Brown and Haas [51] and by Gemulla et al. [48,49].

Classical statistics. Some of the classical column statistics can be quite challenging to estimate from a sample, and a detailed discussion is beyond the present scope. For examples of the complexities involved, see the discussions in [52–54] on estimation of column cardinality (under both row- and page-level sampling), in [55] on estimation of the k most frequent values, and in [56] on estimation of the extreme values. Quantiles are perhaps the easiest of the column statistics to estimate from a sample—quantile estimation is a classical topic in the finite-population sampling literature [57, Sec. 5.11].

Algebraic constraints. The 'bump-hunting' B-HUNT tool of Brown and Haas [58] automatically discovers algebraic constraints between pairs of columns in relational data. These constraints are a variant of the type discussed in [30]—see Section 2.1.1—and take the form $y \oplus x \in S$, where $\oplus \in \{+, -, /\}$ and S is a union of disjoint intervals. In general, the algebraic constraints may be 'fuzzy,' in that they hold for most, but not all, of the records in the database. Figure 6 illustrates a fuzzy constraint that involves a union of three disjoint intervals (i.e. three 'bumps' in the histogram), which holds for all but a small number of records. A key feature of B-HUNT is that the constraint discovery is based on a sample of the data, so that fuzziness is unavoidable in general. Nonetheless, these fuzzy constraints can be taken as strong evidence of correlation, and so can be used to improve selectivity estimation. Moreover, B-HUNT can be used to improve the efficiency of data access during query processing. The idea is to generate special 'exception' tables to store the (small) set of records that violate a discovered constraint. Then the fuzzy constraints can be used to rewrite queries so as to take advantage of indexes, data partitioning, and so forth, in analogy to [30]. The execution of each rewritten query is followed by special processing to correctly handle the data in the exception tables; this latter processing is inexpensive, because the exception tables are small.

Algebraic constraints are discovered by applying statistical histogramming, segmentation, or clustering techniques to the sampled data. Exploiting classic sampling results of Scheffé and Tukey [59,60], B-HUNT automatically selects a sample size that ensures (with high probability) a user-specified upper bound on the size of the exception table, and hence on the cost of exception-table processing. The size and number of bump intervals are chosen to optimally

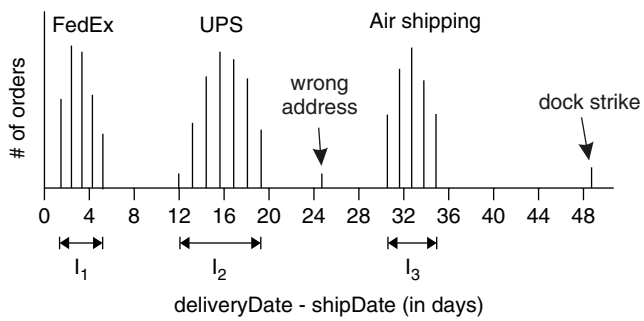

Fig. 6 Example of a fuzzy algebraic constraint: deliveryDate – shipDate $\in I_1 \cup I_2 \cup I_3$.

trade off the filtering power of the constraint and the complexity of processing the rewritten query that exploits the constraint.

B-HUNT employs a novel analysis of the database schema to generate pairs of candidate attributes. The general approach involves systematically enumerating candidate attribute pairs while simultaneously pruning unpromising candidates using a flexible set of heuristics. Since the attributes in a candidate pair can belong to different tables, the enumeration process involves, among other things, finding declared or undeclared key columns and then finding foreign keys that point to these columns (Section 1.2). A ranking of the discovered relationships based on estimated 'benefit' is used to limit the number of maintained algebraic constraints to be used during query optimization. Figure 7 shows the effect of using B-HUNT on the execution times of a set of queries from the TPC-H benchmark, where the database size exceeds 2.3 Tb. As can be seen, there were no significant performance decreases, a number of queries had significant speedups, and one particularly long-running query was sped up by a factor of almost 7.

Correlation discovery. As discussed previously, knowledge of correlations between attributes is useful in avoiding erroneous independence assumptions when estimating selectivities, as well as in determining groups of columns on which to maintain joint statistics such as the CG statistics mentioned in Section 1.2. The CORDS (CORrelation Detection via Sampling) tool of Ilyas et al. [61] automatically discovers correlations and 'soft' functional dependencies between columns. A soft FD between columns C_1 and C_2 is a generalization of the classical notion of a 'hard' FD in which the value of C_1 completely determines the value of C_2. In a soft FD (denoted by $C_1 \Rightarrow C_2$), the value of C_1 determines the value of C_2 with high probability. An example of a hard FD is given by `Country` and `Continent`; the former completely determines the latter. On the other hand, for cars, `Make` is determined by `Model` via a soft dependency: given that `Model` = 323, we know that `Make` = `Mazda` with high probability, but there is also a small chance that `Make` = `BMW`.

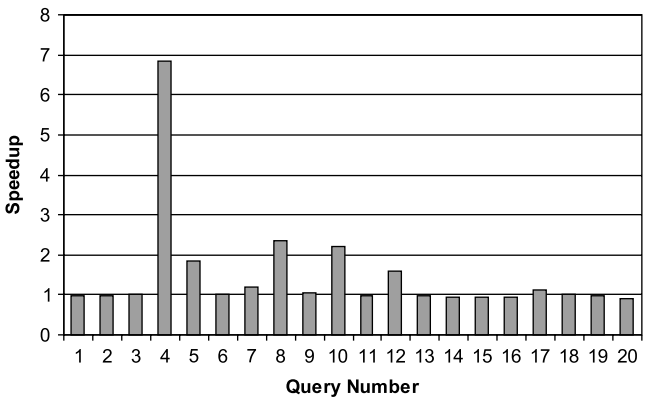

Fig. 7 Effect of B-HUNT on TPC-H query execution times.

CORDS both builds upon and significantly modifies the technology of the B-HUNT system discussed previously. As with B-HUNT, CORDS searches for column pairs that might have interesting and useful correlations in a manner similar to B-HUNT, combining systematic enumeration with heuristic pruning. Also, as with B-HUNT, CORDS analyzes only a sample of rows in order to ensure scalability to very large tables. The similarities end here, however. Although B-HUNT discovers soft algebraic relationships between numerical attributes, CORDS employs a robust chi-squared analysis to identify correlations between both numerical and categorical attributes, and an analysis of the number of distinct values in the sampled columns to detect soft FDs. The chi-squared analysis is 'robust' in the sense that the algorithm automatically checks for trivial cases such as soft keys and single-valued columns, and automatically deals with skew in the data when determining the data grouping for the chi-squared contingency table. The sample size required for the chi-squared analysis is essentially independent of the database size, so that the algorithm is highly scalable.

In slightly more detail, CORDS operates as follows. For a table R, consider two attributes $R.A$ and $R.B$ with domains D_A and D_B, and denote by $f_{\alpha\beta}$ a sampling-based estimate of the fraction of rows r such that $r.A = \alpha$ and $r.B = \beta$. Also denote by $f_{\alpha+}$ and $f_{+\beta}$ the corresponding estimated marginal relative frequencies. A correlation between two columns is roughly defined to be a 'significant' departure from the *independence condition*. This latter condition is defined to hold if and only if $f^{\mathrm{a}}_{\alpha\beta} = f^{\mathrm{a}}_{\alpha+} f^{\mathrm{a}}_{+\beta}$, where f^{a} denotes the actual relative frequency of an attribute value in R. Assume for simplicity that the number of attribute values is small enough, and the attribute-value frequencies are uniform enough, so that no bucketization is needed; i.e. the rows and columns of the two-way contingency table correspond to the distinct values of the respective attributes. (We use standard statistical terminology: a contingency table based on M observations is a display of the attribute-value frequencies in a $D_A \times D_B$ array, with table entries of the form $n_{\alpha\beta} = M f_{\alpha\beta}$; moreover, marginal absolute frequencies of the form $n_{\alpha+} = M f_{\alpha+}$ and $n_{+\beta} = M f_{+\beta}$ are displayed as row and column totals, respectively.) To test for correlation, CORDS uses the standard chi-squared statistic, defined for M observations by

$$\chi^2 = M \sum_{\alpha \in D_A} \sum_{\beta \in D_B} \frac{(f_{\alpha\beta} - f_{\alpha+} f_{+\beta})^2}{f_{\alpha+} f_{+\beta}}.$$

The correlation test rests on the fact that if the attribute frequencies approximately satisfy the independence condition and M is large, then χ^2 will have approximately a

chi-squared distribution with r degrees of freedom, where $r = (|D_A| - 1)(|D_B| - 1)$. CORDS asserts a soft FD of the form $C_1 \Rightarrow C_2$ if and only if the ratio $D(C_1)/D(C_1, C_2)$ is close to 1, where $D(C_1)$ is the number of distinct values in column C_1, and $D(C_1, C_2)$ is the number of distinct (C_1, C_2) value pairs in the table.

Experimental results for the initial CORDS prototype in DB2 for LUW (Fig. 8) indicate that merely discovering two-way correlations—and thus maintaining CG statistics on pairs of columns in DB2—reduced the worst-case selectivity-estimation errors by an order of magnitude for a real-world query workload. Maintaining three-way joint statistics provided some further improvement, but the marginal benefit of maintaining higher-order joint statistics is seen to diminish sharply, especially in the face of the sharply increasing cost of correlation discovery and joint-statistics maintenance. A similar phenomenon is mentioned in [40]. A scatter plot of the query execution time with and without CORDS for 300 real-world queries—see Fig. 9—shows that the worst-case query execution time was also reduced by an order of magnitude.

Because correlation between attributes is so prevalent in the real world, any correlation-detection method is likely to discover a large number of correlated attribute pairs. It is therefore often necessary to rank discovered pairs in decreasing order of correlation; then, for example, multivariate statistics can then be maintained on the 'most correlated' attributes, subject to memory and processing constraints. To obtain a ranking, the χ^2 statistic can be normalized to yield a measure of the 'distance from independence.' This distance is called the *mean-square contingency distance* (MSCD), and is defined in Cramér [62] as $\phi^2 = \chi^2/\big[M(d-1)\big]$, where $d = \min(|D_A|, |D_B|)$. It can be shown that $0 \le \phi^2 \le 1$. Moreover, $\phi^2 = 0$ if and only if independence holds, and $\phi^2 = 1$ if and only if a functional dependency exists between the attributes. The attribute pairs can then be ranked in decreasing order of MSCD.

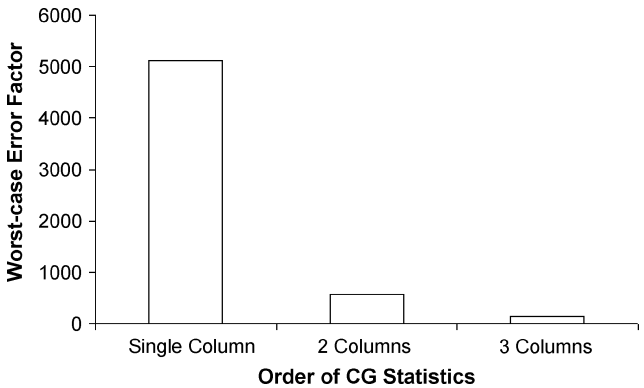

Fig. 8 Effect of column-group order on accuracy of selectivity estimates.

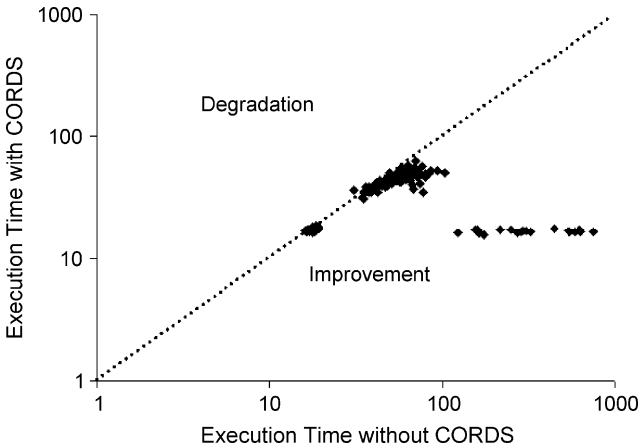

Fig. 9 Effect of CORDS on query execution time.

In the CORDS work and in its extension to a query-feedback setting (see Section 3.1 below), the MSCD was used as the measure of correlation, i.e. 'distance from independence,' mostly for convenience, because the metric is relatively simple and its statistical properties are well understood. In principle, CORDS and related methods can be based on other measures of statistical dependence. See, for example, Read and Cressie [63] for a general family of association measures based on 'power divergence' statistics. This family includes MSCD and mutual information as special cases; the different members of the family are sensitive to different types of departures from independence, and the authors make a case for a member of the family that does not correspond to the usual metrics.

2.2. Workload-aware Methods

The approaches described in Section 2.1 proactively analyze the database schema and the data items themselves to discover useful statistical information. However, knowing more about the expected query workload gives richer context and allows targeted, efficient discovery of useful statistical relationships. In the following, we summarize a variety of proactive, workload-aware approaches.

Selecting the most important statistics. Knowledge of the workload can be exploited to concentrate the limited resources available for statistics collection and storage on the 'hottest' portions of the database, i.e. those most likely to enjoy future activity. One tool for doing this is the Statistics Advisor, which is part of the Optimizer Expert tool in DB2 for z/OS; see Bruni et al. [64]. This tool analyzes the predicates in all queries of the workload to determine which statistics to collect, and generates specific RUNSTATS statements appropriately. For example, RUNSTATS will collect a relatively large number of frequent-value statistics on a column that is referenced in many equality predicates. If

the workload is available before it needs to be run, then the statistics can be collected in time to benefit optimization of that workload's queries. Otherwise, the statistics, once collected, will benefit later executions of the same or similar workload.

Gathering statistics at optimization time. A number of researchers have suggested collecting statistics at the time a query is optimized, thereby exploiting knowledge about the query to guide the statistics collection. For Microsoft SQLServer, Chaudhuri and Narasayya [65] have prototyped an approach, called Magic Number Sensitivity Analysis (MNSA), that attempts to leverage the query optimizer in order to determine the statistics that are most relevant to a given workload query, and hence should be proactively collected prior to optimization. The optimizer is invoked twice in order to decide whether the current set of statistics is sufficient. In the first invocation, all unknown selectivities are set to a very small value $\epsilon > 0$ and in the second invocation, all unknown selectivities are set to a large value $1 - \epsilon$. If the estimated costs of the two QEPs are within $t\%$ of each other (for a predefined 'magic number' t), the current set of statistics is considered sufficient, otherwise the system identifies the most important statistics to collect. These statistics are identified by calling the optimizer again to get a QEP based on the current set of statistics, together with the estimated costs for each operator in the plan. The statistics associated with the most expensive operators are designated as the most important.

More recently, El-Helw et al. [66], in developing their just-in-time statistics (JITS) framework, propose an alternative proactive approach to determine, collect, and materialize statistics just prior to the optimization of each query. The statistics take the form of selectivities for predicates in the query. The goal is not only to improve the selectivity estimation for the query at hand, but to exploit the statistics when optimizing 'similar' queries in the future.

Figure 10 gives a high-level architectural description of JITS. Entities drawn with dotted lines already exist in current query engines, whereas entities in solid lines are new JITS modules. The *QSS archive* is a repository of adaptive single- and multidimensional histograms that incorporate the collected query-specific statistics. Specifically, the system may decide to integrate a newly collected statistic into the current set of histograms. The integration is performed using maximum-entropy techniques; see the discussion of the ISOMER histogram in Section 3.2 for further details. The *Query Analysis* module analyzes the query structure, after parsing and rewriting, to determine all relevant statistics, and generates a list of candidate statistics to collect. The *Sensitivity Analysis* module processes the candidate statistics to decide the most crucial statistics to collect by examining the query and the existing statistics, along with the history of data activity, e.g. the frequency of updates and deletes on a particular table. The sensitivity analysis module can potentially exploit sophisticated techniques to determine the sensitivity of the query to a particular statistic, e.g. by incorporating the planning module in a manner similar to the MNSA method in [65]. The *Statistics Collection* module collects the required statistics, and uses them to update the QSS archive of histograms. The *Plan Generation and Costing* module uses the information in the QSS archive and the system catalog to select a QEP. Finally, the *Statistics Migration* module periodically updates the system catalog using the information in the QSS archive.

A key difference between JITS and the scheme in [65] is that JITS uses a much lighter-weight sensitivity analysis than MNSA, avoiding expensive optimizer invocations. Such invocations can also be inaccurate when statistics are badly out of date, because the QEPs from which costs are computed are based on inaccurate information. Another key difference is that the statistics collected by JITS can be query-specific.

Statistics on intermediate tables. For SQLServer, Bruno and Chaudhuri [67] extend the above approaches and develop a method called Statistics on Intermediate Tables (SITS) that collects and stores statistics not just on base tables, but on the result of queries. SITS encapsulate important statistical information about a dataset. In DB2 for LUW [68], SITS are also called 'statistical views'—cf. Section 1.2. SITS or statistical views can be obtained proactively by running a utility such as RUNSTATS on the result of the defining query. The major challenges of this approach are: (i) determining for which of the many subexpressions of an SQL workload the system should collect SITS, and (ii) ensuring that the query optimizer is able to exploit SITS if they exist for some subexpression of a given query. Bruno and Chaudhuri use MNSA sensitivity analysis to solve challenge (i), and standard view-matching algorithms to handle challenge (ii). Experimental results in [67] showed improvements in query time up to two orders of magnitude.

3. REACTIVE APPROACHES

The methods in the previous section required proactive data gathering, either by the user or by the system, to discover statistical structure in the data. In recent years, increasing attention has been given to approaches that piggyback on top of query execution. The motivation is simple: since information is being gathered when processing production queries, why not exploit this 'free' information? Of course, this approach will only work if the information truly is 'free,' or almost so. A key concern is the overhead of monitoring the query execution and recording the

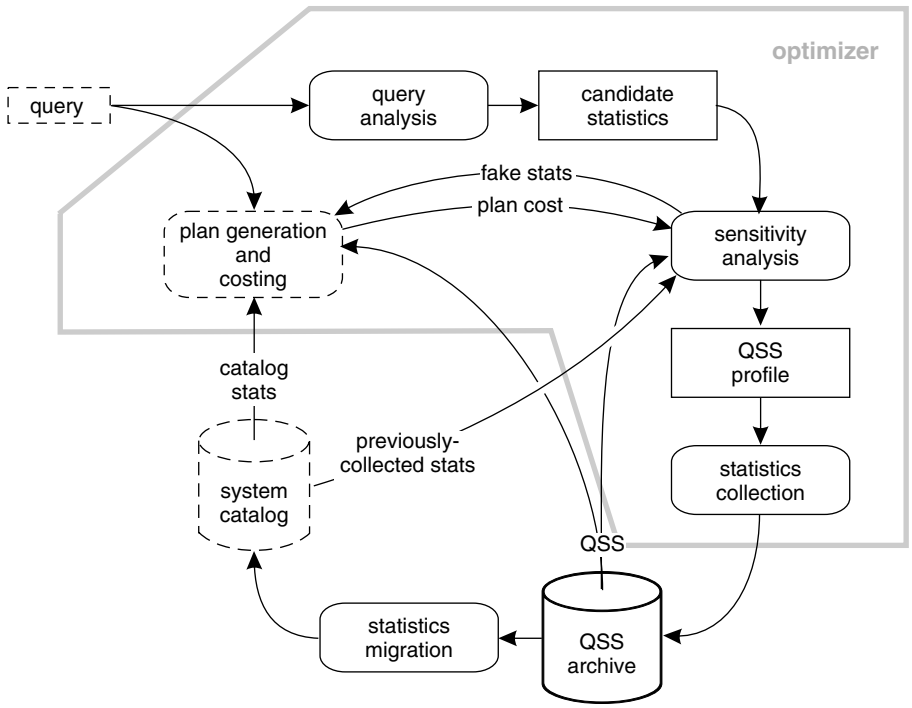

Fig. 10 JITS architecture.

statistics. The initial version of the LEO (LEarning Optimizer) project [69] opportunistically gathered statistics on a view as a byproduct of query execution, with the goal of computing multiplicative 'correction factors' for the optimizer's selectivity estimates. Experiments with the LEO prototype indicated that the required monitoring could be accomplished with an acceptably small runtime overhead (typically less than 5%).

The LEO approach has evolved into a method for automated statistics configuration; see Aboulnaga et al. [70]. As each predicate is evaluated during query execution, a *query feedback record* (QFR) is created that comprises a predicate descriptor, the actual predicate cardinality, and the optimizer's cardinality estimate. LEO populates a *query feedback warehouse* (QFW) with these QFRs and uses them for a variety of statistics-configuration tasks, including the discovery of correlations, the automatic triggering of statistics collection, and determining the number of quantiles and frequent values to collect on each column. The remainder of this section describes these, and other, reactive methods.

3.1. Detecting Correlations

We now describe several feedback-based analogs to the proactive correlation-discovery methods described in Section 2. The SASH method of Lim et al. [71] adapts the graphical statistical modeling approach in [40]—see Section 2.1.1—to the reactive setting. The 'restructuring phase' of SASH uses feedback to build a decomposable interaction model, just as in [40], to represent the correlation structure of the data. That is, the attributes in a table are partitioned into cliques, a multidimensional histogram is maintained for each clique, and joint selectivities for attributes in different cliques are obtained by combining marginal selectivities according to rules specified by the interaction model. A key difference from [40] is that a candidate model is scored by comparing the difference between the true selectivities observed in a set of feedback queries and the selectivities estimated from the candidate model. The search algorithm is also slightly different: at each step, the next 'move' is determined as either a greedy local improvement in the assignment of attributes to cliques or a greedy improvement in the configuration of bins in a histogram for one of the existing cliques, whichever yields the largest decrease in estimation error relative to the feedback queries. Finally, the histograms are updated using a 'delta' (steepest descent) rule. As discussed below, this updating approach can lead to inaccuracies, because feedback is not added in a consistent manner. In principle, a consistent, maximum-entropy-based approach (as described in Section 3.2 below) can be used instead, at the cost of increased complexity. As with the proactive version of this approach, the complexity and cost of the method has so far prevented it from being incorporated into commercial systems. Moreover, there are a number of applications, such as database compression [72] and analysis of data from

system monitors, where it suffices simply to discover sets of correlated attributes, but the foregoing methods do not allow such discovery to be decoupled from the task of constructing detailed histograms.

At the opposite end of the spectrum, an extremely simple approach was the *correlation analyzer* (CA) method proposed in Aboulnaga et al. [70], which detects pairwise correlations. As in Section 2.1.2, consider two attributes $R.A$ and $R.B$ with domains D_A and D_B, and now denote by $f_{\alpha\beta}$ the actual fraction of rows such that $r.A = \alpha$ and $r.B = \beta$. Also denote by $f_{\alpha+}$ and $f_{+\beta}$ the corresponding (actual) marginal relative frequencies. The QFW provides a set of observations $\mathcal{O} = \{\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_n\}$, where $1 \le n < |D_A \times D_B|$ and each observation $\mathcal{O}_i$ concerns a conjunctive predicate of the form '$A = \alpha_i$ AND $B = \beta_i$' with $\alpha_i \in D_A$ and $\beta_i \in D_B$. Each subpredicate that appears in the conjunctive predicate, e.g. '$A = \alpha_i$' in the above example, is called a *simple* predicate. Each observation $\mathcal{O}_i$ is a set having one of the forms

(i) $\mathcal{O}_i = \{ f_{\alpha_i\beta_i}, f_{\alpha_i+}, f_{+\beta_i} \}$
(ii) $\mathcal{O}_i = \{ f_{\alpha_i\beta_i}, f_{\alpha_i+} \}$
(iii) $\mathcal{O}_i = \{ f_{\alpha_i\beta_i}, f_{+\beta_i} \}$
(iv) $\mathcal{O}_i = \{ f_{\alpha_i\beta_i} \}$

depending on the feedback that is available. Then, for a particular observation $\mathcal{O}_i = \{ f_{\alpha_i\beta_i}, f_{\alpha_i+}, f_{+\beta_i} \}$, an observational correlation is declared if the relationship

$$1 - \epsilon \le \frac{f_{\alpha_i\beta_i}}{f_{\alpha_i+} f_{+\beta_i}} \le 1 + \epsilon,$$

fails to hold, where ϵ is a small prespecified positive parameter less than 1. If a pair of attributes has at least two observational correlations, then the attributes are considered correlated. The original description in [70] does not specify what to do if f_{α_i+} or $f_{+\beta_i}$ is unavailable. In such a case, the method proposed in Haas et al. [73] can be used—the idea is to choose the missing values to provide as much support for the independence as possible, while being roughly consistent with the optimizer estimates (i.e. staying within the observed degree of error in the QFW). This choice of missing values minimizes the chance of incorrectly discovering a correlation; such false positives are to be avoided because they lead to significant storage and maintenance costs. For purposes of correlation ranking—cf. the discussion of CORDS—the CA method computes the correlation measure for the attribute pair (A, B) essentially as $\sum_i |f_{\alpha_i\beta_i} - f_{\alpha_i+} f_{+\beta_i}|$. The problem with the CA method is that the value of ϵ is completely ad hoc. Moreover, the CA method does not combine all feedback observations pertinent to attributes A and B in a principled manner, which can lead to unstable behavior of the detection algorithm. Finally, the dependency measure has been shown experimentally to lead to unstable, erratic rankings.

To address these problems, a principled yet relatively simple method for correlation detection and ranking was recently introduced by Haas et al. [73]. The idea is to emulate the chi-squared analysis of CORDS. A key technical challenge in the reactive setting is that frequency information is not available for all attribute-value pairs $(\alpha, \beta) \in D_A \times D_B$, as required for chi-squared theory. Information is at hand only for those pairs $\{(\alpha_i, \beta_i) : 1 \le i \le n\}$ that correspond to the available feedback observations; i.e. the contingency table is incomplete.

To deal with this technical issue, the method in [73] uses a statistic of the form $H_M = Mx^{\mathrm{t}}Qx$, where the superscript 't' denotes transpose, the vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is defined by

$$x_i = \frac{f_{\alpha_i\beta_i} - f_{\alpha_i+} f_{+\beta_i}}{f_{\alpha_i+} f_{+\beta_i}},$$

for $1 \le i \le n$ (where we take $0/0 = 1$), and the matrix Q is specified as the pseudoinverse of the $n \times n$ matrix $\Sigma = \|\sigma_{ij}\|$ given by

$$\sigma_{ij} = \begin{cases} \frac{(1-f_{\alpha_i+})(1-f_{+\beta_i})}{f_{\alpha_i+} f_{+\beta_i}} & \text{if } i = j; \\ -\frac{1-f_{\alpha_i+}}{f_{\alpha_i+}} & \text{if } i \ne j,\ \alpha_i = \alpha_j, \text{ and } \beta_i \ne \beta_j; \\ -\frac{1-f_{+\beta_i}}{f_{+\beta_i}} & \text{if } i \ne j,\ \alpha_i \ne \alpha_j, \text{ and } \beta_i = \beta_j; \\ 1 & \text{if } i \ne j,\ \alpha_i \ne \alpha_j, \text{ and } \beta_i \ne \beta_j. \end{cases}$$

It can be shown that $H_M = 0$ if and only if the independence condition holds. Moreover, for large M, the statistic H_M has approximately a chi-squared distribution with r degrees of freedom. Here r is the rank of Q and a superpopulation model is assumed in which rows of R are generated from a joint probability distribution p on (D_A, D_B), where $p_{\alpha\beta} = P\{r.A = \alpha \text{ and } r.B = \beta\} = f_{\alpha+} f_{+\beta}$ for each $\alpha \in D_A$ and $\beta \in D_B$. That is, H_M has roughly a chi-squared distribution when the table is generated from a truly independent distribution whose marginals coincide with those actually observed in the data. Using these results, correlation detection can proceed in a manner almost identical to CORDS. The above development has assumed complete feedback observations; incomplete observations are handled as described previously. For n QFRs, the complexity of the H_M computation is $O(n^3)$, which is expensive, but the method can be made practically feasible by using a random sample of QFRs and/or by incrementally maintaining H_M using singular-value decomposition (SVD) updating techniques; see [73].

Because the contingency table is incomplete in the reactive setting, ranking correlated attribute pairs is also more

challenging than in the proactive CORDS setting. The MSCD cannot be computed, and normalization becomes nontrivial. As shown in [73], an upper bound on H_M can be obtained via an application of the Courant-Fischer Minimax Theorem, and H_M can be divided by this bound to scale it to the interval $[0, 1]$. This bound, however, can be numerically unstable. Fortunately, experiments in [73] indicate that, for ranking purposes, it suffices to approximate the upper bound by a high (e.g. 0.99) quantile of the chi-squared distribution.

3.2. Maintaining Frequency Distributions

As can be seen from the foregoing discussion, maintaining a univariate and/or multivariate histogram of attribute frequencies can be a key task, both directly in query optimization and indirectly in discovering correlation structure. Proactive methods for histogram construction are based on sampling [16,20,74] or scanning [10,21–23,75]. In this section we show how query feedback can be used not only to determine which histograms to maintain, as discussed in Section 3.1, but also to incrementally maintain an approximation for a frequency distribution by adjusting the buckets of a histogram $\mathcal{H}(A_1, \ldots, A_d)$ on attributes $A_1, A_2, \ldots, A_d$ based on observed cardinalities. In the following, we assume for ease of exposition that each attribute A_i is numerical, taking values in an interval $[l_i, u_i]$, so that the joint frequency distribution of the data is defined over the Cartesian product space $\mathcal{S} = \prod_{i=1}^{d} [l_i, u_i]$.

Incremental maintenance of a histogram based on query feedback has to address the following challenges:

1. *Enforcing consistency*: The histogram distribution must, at every time point, be consistent with all currently valid feedback and not incorporate any ad hoc assumptions.
2. *Dealing with data changes*: Changes of the databases through updates, inserts, and deletes can invalidate old query feedback. Such feedback must be efficiently identified and its effect on the histogram must be undone.
3. *Meeting a limited space budget*: An RDBMS usually limits the size of a histogram to a few disk pages. Adding more feedback to the histogram while maintaining consistency leads to an increase in the histogram size. To meet a limited space budget, the relatively 'important' feedback must be identified and retained, and the less important feedback discarded.

The notion of using feedback for estimating selectivities was originally proposed by Chen and Roussopoulos [43]. Pioneering work on adaptive histograms is described in [76,77], and, as discussed previously, the SASH algorithm in [71] contains an adaptive histogram mechanism. In particular, Bruno et al. [77] introduced a novel data structure called STHOLES. This data structure—in which buckets can have recursively defined holes—achieves high storage efficiency for a histogram in the presence of highly nonuniform attribute-value distributions (Fig. 11). A significant deficiency in all of this work is a lack of the crucial consistency property, even for static data. The ISOMER method of Srivastava et al. [78] addresses all of the above challenges by using the STHOLES data structure in combination with a maximum-entropy principle, approximating the true data distribution by the 'simplest' distribution that is consistent with all of the currently valid feedback.

The idea behind ISOMER is to initially use a single histogram bucket, so that all attribute values are assumed to be mutually independent and uniformly distributed. Given a piece of query feedback, i.e. given the number of rows that satisfy a specified predicate q, ISOMER first adjusts bucket boundaries (possibly creating new buckets in the process) so that each bucket lies either completely inside or completely outside the region in attribute-space specified by q; Fig. 12 illustrates the process in two dimensions. ISOMER then uses the maximum-entropy principle to adjust the bucket counts so that the data distribution represented by the histogram is the 'simplest' distribution that is consistent with the feedback information observed so far.

In more detail, after a boundary adjustment, denoted by $b_1, b_2, \ldots, b_k$ the current set of buckets in the multidimensional ISOMER histogram, and by $n(b_1), n(b_2), \ldots, n(b_k)$ the corresponding bucket counts. Each bucket b_i has a hyperrectangular bounding box denoted by $C(b_i)$; i.e. bucket b_i is bounded between two constant values in each dimension. (Although ISOMER actually uses the STHOLES data structure, in the following discussion we simplify the exposition by ignoring the fact that holes can occur in buckets, because this detail only affects the formula for the volume of a bucket; see [78] for a discussion in full generality). We also denote by $C(q)$ the hyperrectangular subregion of $\mathcal{S}$ that is specified by predicate q. The boundary adjustment step ensures that, for each bucket b and predicate q_i incorporated so far, either $C(b) \subseteq C(q_i)$

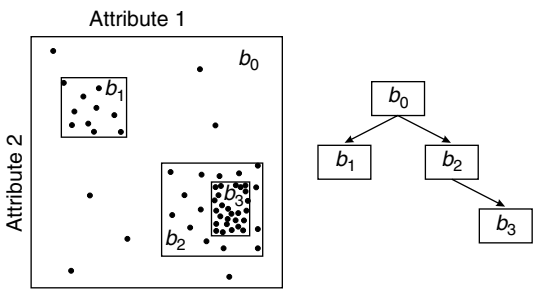

Fig. 11 STHOLES data structure (two dimensions).

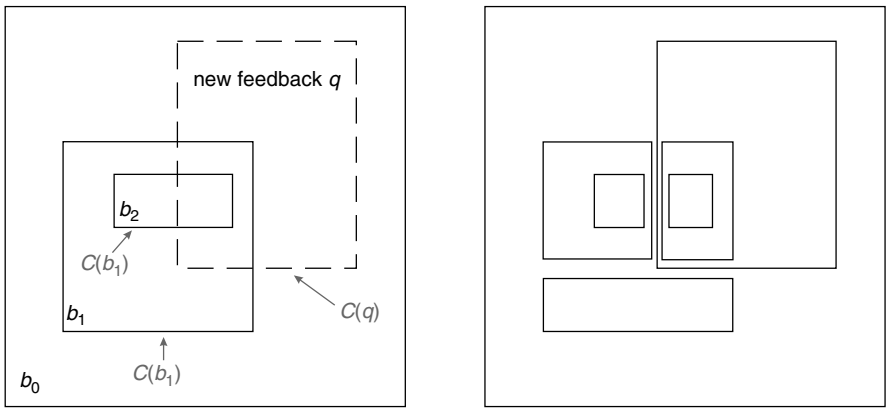

Fig. 12 Bucket-boundary adjustment in ISOMER.

or $C(b) \cap C(q_i) = \emptyset$. ISOMER uses the query feedback to impose constraints on the permissible bucket counts of a histogram, and solves the problem of incorporating new query feedback as a constrained entropy maximization problem over the bucket counts. That is, ISOMER associates a probability distribution $\mathcal{P}$, and hence an entropy value $H(\mathcal{P})$, with every possible histogram over the current set of buckets. In accordance with the maximum-entropy principle, ISOMER then maximizes $H(\mathcal{P})$ over the set of all such histograms that are consistent with the current set of constraints from the query feedback.

To define $\mathcal{P}$ and $H(\mathcal{P})$, consider an arbitrary histogram having k Buckets, denoted by D the number of distinct tuple values (i.e. distinct elements of $\mathcal{S}$) that occur in the table and by $\mathcal{D} = \{ v_1, v_2, \ldots, v_D \}$ the set of these values. The probability distribution associated with the histogram is $\mathcal{P} = (p_1, p_2, \ldots, p_D)$, where p_j is the probability that a tuple randomly selected from the table has value equal to v_j. To obtain an explicit formula for p_j, note that ISOMER assumes that the D distinct values are distributed uniformly throughout $\mathcal{S}$, so that a fraction $f(b) = \mathrm{vol}\big[C(b)\big]/\,\mathrm{vol}(\mathcal{S})$ of these values lie in bucket b. There are $n(b)$ tuples in bucket b, and ISOMER assumes a uniform distribution among the distinct values in a bucket. Denote by b_j^* the bucket in which the value v_j lies. The estimated probability that a randomly selected tuple has value v_j is the probability that the tuple belongs to b_j^* times the probability that the value of the tuple is equal to v_j and not to any of the other distinct values in bucket b_j^*. The first probability equals $n(b_j^*)/N$ and the second probability equals $1/\big[f(b_j^*)D\big]$, where N is the total number of rows in the table. Thus $p_j = n(b_j^*)/\big[f(b_j^*)ND\big]$ for $1 \le j \le D$. The entropy $H(\mathcal{P})$ corresponding to the distribution $\mathcal{P}$ is therefore given by

$$\begin{aligned} H(\mathcal{P}) &= -\sum_{j=1}^{D} p_j \ln(p_j) \\ &= -\sum_{j=1}^{D} \frac{n(b_j^*)}{f(b_j^*)ND} \ln\left[\frac{n(b_j^*)}{f(b_j^*)ND}\right] \\ &= -\sum_{i=1}^{k} \sum_{j|v_j \in C(b_i)} \frac{n(b_i)}{f(b_i)ND} \ln\left[\frac{n(b_i)}{f(b_i)ND}\right]. \end{aligned}$$

For each value of i, since bucket b_i has $f(b_i)D$ distinct values, the inner sum consists of $f(b_i)D$ identical terms. Using this observation together with the fact that $\sum_{i=1}^{k} n(b_i) = N$, we find that

$$\begin{aligned} H(\mathcal{P}) &= -\sum_{i=1}^{k} \frac{n(b_i)}{N} \ln\left[\frac{\mathrm{vol}(\mathcal{S})n(b_i)}{\mathrm{vol}\big(C(b_i)\big)ND}\right] \\ &= -\frac{1}{N}\sum_{i=1}^{k} n(b_i) \ln\left[\frac{n(b_i)}{\mathrm{vol}\big(C(b_i)\big)}\right] - \ln\left[\frac{\mathrm{vol}(\mathcal{S})}{ND}\right]. \end{aligned}$$

The query feedback leads directly to a set of constraints on the histogram bucket counts. Suppose that we have obtained feedback for m predicates $q_1, q_2, \ldots, q_m$. Recall that, by construction, either $C(b) \subseteq C(q_i)$ or $C(b) \cap C(q_i) = \emptyset$ for each bucket b and predicate q_i. It follows that the constraint corresponding to predicate q_i can be written as

$$\sum_{b|C(b)\subseteq C(q_i)} n(b) = N(q_i), \tag{1}$$

where $N(q_i)$ is the number of tuples that satisfy q_i, as determined from the query feedback. This constrained optimization problem can be solved efficiently using, e.g. an

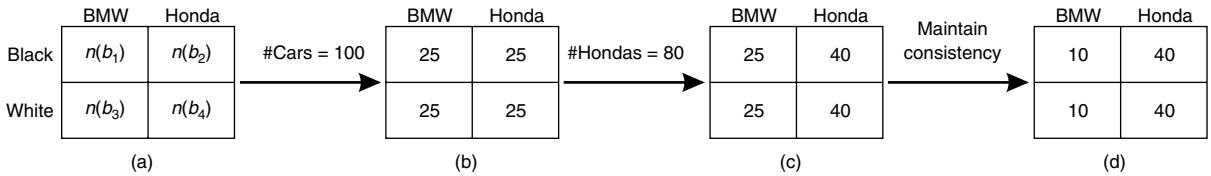

Fig. 13 Maximum entropy estimation in ISOMER.

'iterative scaling' algorithm [78]. Figure 13 illustrates, for a fixed bucket configuration, how the estimates are updated as new feedback arrives. Note that the naive update in Fig. 13(c) preserves the uniformity between black and white cars and respects the feedback '#Hondas 80', but has violated the original feedback of '#Cars = 100.' After the maximum-entropy adjustment, the distribution in Fig. 13(d) respects both pieces of feedback, and the distribution is as uniform as possible.

Updates to a database may make the constraint sets inconsistent, by causing some of the constraints (especially older constraints) to no longer hold. In order to ensure consistency in the presence of updates, ISOMER detects inconsistencies using a linear programming approach that associates two 'slack' variables with each constraint. The constraints in Eq. (1) are rewritten as

$$\sum_{b|C(b)\subseteq C(q_i)} n(b) - N(q_i) = s_i^+ - s_i^-, \qquad (2)$$

for $1 \le i \le m$. In addition, ISOMER adds the nonnegativity constraints

$$n(b) \ge 0 \text{ for all } b, \quad s_i^+, s_i^- \ge 0 \text{ for } i = 1, \ldots, m. \qquad (3)$$

If there is a solution to the set of constraints Eqs (2) and (3) such that $s_i^+ = s_i^- = 0$, then the solution satisfies the ith constraint from Eq. (1). Alternatively, if s_i^+ or s_i^- is positive, then the solution does not satisfy this constraint. Ideally, we would like a solution that satisfies the maximum number of constraints from Eq. (1), i.e. a solution that minimizes the number of nonzero slack variables. Unfortunately, the problem of determining such a maximum satisfiable subset from a set of constraints is known to be NP-complete [79]. ISOMER instead settles for minimizing the sum of the slack variables, because this problem can be efficiently solved using linear programming methods. ISOMER then discards all constraints having nonzero slack. The actual ISOMER approach refines the above problem formulation by adding weights to the constraints, in order to discard older, outdated QFRs rather than more recent feedback; see [78] for details.

An elegant aspect of ISOMER is that the maximum-entropy solution yields, for free, an importance measure for each constraint. ISOMER exploits this fact to obtain a very efficient procedure for detecting and discarding unimportant constraints, in order to meet a limited space budget. Specifically, ISOMER uses the quantity $|\lambda_i|$ as the importance measure for the ith QFR, where λ_i is the Lagrange multiplier associated with the ith constraint in the entropy maximization problem. To justify this choice intuitively, we note that if $\lambda_i = 0$, then removal of the ith feedback record does not affect the bucket counts, so that the final maximum-entropy solution is unaltered. Figure 14 illustrates the improvement in selectivity estimates obtained by a 3D ISOMER histogram relative to both the standard DB2 optimizer and an algorithm called STGRID that uses the STHOLES data structure but updates the histogram in a naive manner—as in Fig. 13(c)—rather than applying the maximum entropy principle.

Although the full ISOMER multidimensional histogram has not yet been implemented in a commercial system, a one-dimensional version has been prototyped for inclusion into the informix dynamic server (IDS) product—see Behm et al. [80]—to improve selectivity estimates for single columns. This industrial-strength version exploits the one-dimensional nature of the histogram in order to simplify several aspects of ISOMER. First, the complicated

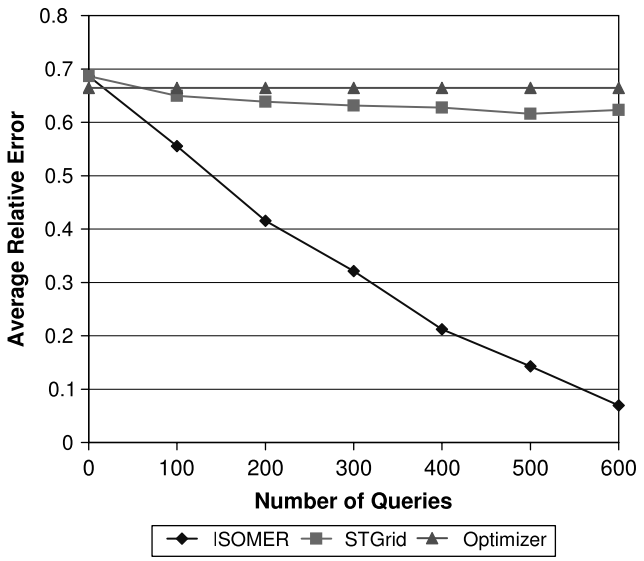

Fig. 14 Accuracy of selectivity estimates.

STHOLES data structure is replaced with a simple bucketization scheme, using a sweep-line algorithm to determine bucket boundaries. Next, a more efficient method is used to incorporate new feedback into the histogram; the idea is to aggregate, prior to the maximum-entropy computation, those buckets that do not overlap with the new feedback records. Finally, a simple and fast pruning method is introduced to ensure that the number of buckets stays below a specified upper bound.

3.3. Configuring Single-column Statistics

As discussed previously, virtually all commercial RDBMSs maintain, at a minimum, single-column statistics that include distributional information in the form of quantiles and frequent values. A major practical issue is how many quantiles and frequent values to maintain on a given column. Traditionally, the DBA would need to make this decision based on a painful process of trial and error; recent years have seen several attempts by database designers to develop methods for automatically determining the optimal number of these statistics to maintain.

The simple-predicate analyzer (SPA) in DB2 for LUW, described in [70], automatically determines the number of frequent values to maintain on a column. SPA periodically scans the QFW and examines all of the errors in the simple equality predicates that reference the column, to check whether enough frequent values are being maintained in the system catalog. (Because some of the statistics in the catalog are collected using random-sampling techniques, SPA considers only those QFW entries where the observed error exceeds the expected error from normal sampling fluctuations.) If more frequent values are needed, then the SPA automatically recommends an appropriate number of frequent values to maintain. Note that following such a recommendation also results in bringing the frequent-value statistics up to date.

To determine the proper number of frequent values, SPA first scans the QFW and the system catalog to compile a list of all 'known' value frequencies for the column. These include:

1. The frequencies $g_1, g_2, \ldots, g_n$ of the currently maintained frequent values, as recorded in the system catalog,
2. The frequencies $h_1, h_2, \ldots, h_m$ of all values for which there is a relevant record in the QFW. These values can be considered as candidate frequent values to maintain.
3. An average frequency assigned to each of the remaining 'rare' (i.e. infrequent) values. This frequency is computed, using a uniformity assumption, from the estimated number of rows in the table and the number of distinct values in the column.

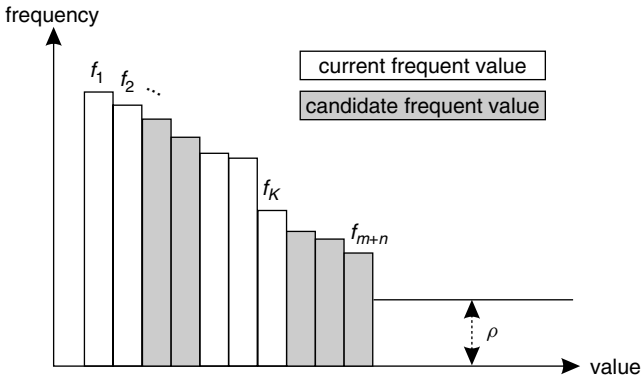

Fig. 15 Frequencies used by SPA.

When multiple frequency estimates are available for a given column value, SPA uses the most recent one. Figure 15 illustrates the frequency list as a bar graph, in descending order of frequency. Suppose that the table has D distinct values in total, and a total cardinality of N rows. Then the successive bar heights are $f_1, f_2, \ldots, f_{m+n}, \rho, \rho, \ldots, \rho$. Here $f_1, f_2, \ldots, f_{m+n}$ are the frequencies $g_1, g_2, \ldots, g_n, h_1, h_2, \ldots, h_m$ arranged in descending order, and ρ is the frequency of the 'rare' values (there are $D - m - n$ of them), defined as

$$\rho = \frac{N - f_1 - f_2 - \cdots - f_{m+n}}{D - m - n}.$$

Based on this 'known' distribution, SPA determines the number K of frequent values to maintain, where $n \leq K \leq m + n$. If K frequent values are maintained, then, when estimating cardinalities, the optimizer uses the exact count for these values and an average count of

$$\overline{f} = \frac{N - f_1 - f_2 - \cdots - f_K}{D - K}.$$

for each of the remaining values. The total absolute optimizer estimation error over all possible simple equality predicates (with respect to the 'known' distribution) is then roughly equal to

$$E(K) = \sum_{i=K+1}^{m+n} |f_i - \overline{f}| + (D - m - n)|\rho - \overline{f}|.$$

The first term represents the contribution due to the $m + n - K$ known frequencies that SPA chooses not to retain, and the second term is the contribution from the remaining values. Observe that $E(K)$ is decreasing in K. To determine the number of frequent values to maintain, SPA initially set $K = n$ (the current number of maintained frequent values) and then increases K until either $E(K)$ falls below

a specified threshold or $K = \min(m + n, B)$, where B is a user-specified upper bound on the number of frequent values to maintain.

A more accurate approach, which we will call SPA+, is described in Behm [81], in which the optimal number of both frequent values and quantiles are determined simultaneously. SPA+ uses query feedback, together with catalog statistics, to maintain an adaptive histogram, using maximum-entropy techniques as described in the previous section. Then, for a given number n of frequent values and m of quantiles, an error $E(n, m)$ can be calculated, e.g. as the Kolmogorov distance between the maximum-entropy distribution (treated here as the 'true' data distribution) and the optimizer's distribution, which is based on n frequent values and m quantiles (The Kolmogorov distance between cumulative distribution functions G and H is defined as $\sup_x |G(x) - H(x)|$ and, in our setting, serves as an upper bound on the absolute error of selectivity estimates over all possible range predicates of the form '$l \leq R.A \leq u$'.) SPA+ starts with the current value (n^*, m^*) and searches through a neighborhood of (n, m) points to seek a better choice of n and m. The size of the search space is specified by the user, and possible search methods include brute-force enumeration and steepest descent. In a sense, SPA+ is dominated by the method in Behm et al. [80], where the relatively accurate maximum-entropy distribution is used directly for selectivity estimation, rather than merely for statistics configuration. On the other hand, SPA+ can be incorporated into current systems relatively easily, without requiring major changes to the optimizer.

3.4. Dynamic Query Reoptimization

Perhaps the most extreme version of a reactive approach to gathering statistical information for query optimization is to collect the information during the execution of the query itself. This dynamically obtained information can then be used to reoptimize the query as it is being processed. This approach makes query processing more robust, and substantially reduces the need for DBA intervention to debug problem queries. Recent approaches to dynamic query reoptimization include the RIO system of Babu et al. [42] and the Progressive Query Optimization (POP) approach described in Markl et al. [82]. Both systems monitor errors in cardinality estimation as intermediate steps of the QEP are executed, in order to decide on how to continue query evaluation.

To avoid thrashing due to continual, expensive reoptimizations, RIO develops 'switchable' plans that can dynamically switch back and forth between a small number of fixed alternative subplans when cardinality estimation errors are detected. Such switching is much cheaper than executing a complete reoptimization. The idea is to represent the uncertainty in a cardinality estimate as an interval around the current estimate; for each operator, up to three alternative subplans are developed under assumptions of minimum, maximum, and currently estimated cardinalities for the operator inputs. During query execution, RIO monitors the actual cardinality values and chooses one of the subplans of a given switchable plan if the monitored values are 'close' to ones used to generate that subplan; if there is no suitable subplan, the query is reoptimized. A drawback of the RIO approach is that RIO cannot always form a switchable plan. Moreover, by limiting the number of possible subplans to three, RIO might miss some possibly good alternative plans. Finally, the RIO approach also depends on the ability to produce meaningful intervals around estimated cardinality, a hard problem in its own right.

POP compares actual and estimated cardinalities at certain natural 'checkpoints' in the QEP. If a large discrepancy is detected at a checkpoint, then query execution is stopped and the query is reoptimized. Oscillation between optimization and execution steps can occur any number of times during a single query execution. A reoptimization step in POP exploits not only the actual cardinalities observed during query execution so far, but also the partial results that have been computed prior to the checkpoint. In this way, POP avoids costly recomputation of already computed results. The mixing of measured cardinalities with statistics-based estimates presents challenging problems for the cardinality model of the query optimizer. POP therefore uses maximum-entropy techniques to ensure that all knowledge is used in a consistent way—see Section 4.1 below—so that progressive reoptimization will not get stuck in an infinite loop.

4. EXPLOITING STATISTICAL INFORMATION

In this section we discuss two practical challenges that arise when trying to exploit discovered statistical information. One important issue is how to deal with inconsistent information, and the other is how to maintain statistical information over time in a heavily utilized system.

4.1. Dealing with Inconsistencies

As discussed in Section 1, inconsistent statistical information can arise when multiple forms of discovery are used simultaneously. For example, there may be 'thin veneer' information in the system catalog that has been obtained by a RUNSTATS-like utility, perhaps augmented by frequency information maintained in a set of ISOMER-style adaptive histograms (over possibly intersecting sets of attributes), perhaps further augmented with information in a QFW. Exacerbating this problem is that the information may have been obtained at different times, and there may have been

updates to the underlying tables that render some of the information obsolete. A net effect of this inconsistency is the possibility that multiple, nonequivalent selectivity estimates may be available for a given predicate.

Specifically, consider the problem of estimating the selectivity $s_{1,2,\ldots,n}$ of a conjunctive predicate of the form $p_1 \wedge p_2 \wedge \ldots \wedge p_n$, where each p_i is a simple predicate of the form '*column op literal*.' Here *column* is a column name, *op* is a relational comparison operator such as ' =,' '>,' or 'LIKE,' and *literal* is a literal in the domain of the column; e.g. 'MAKE = Honda' or 'YEAR > 1984'. An optimizer will always have an estimate of individual selectivities $s_1, s_2, \ldots, s_n$ from the statistics on individual columns, and will typically use independence assumptions in the absence of correlation information. Suppose, however, that $n = 3$ and that joint selectivity estimates $s_{1,2}$ and $s_{2,3}$ are also available, due to discovered correlations between these pairs of attributes. Then possible estimates for $s_{1,2,3}$ include $s_1 s_2 s_3$, $s_{1,2} s_3$, and $s_1 s_{2,3}$, and it is likely that none of these estimates will agree with the others. Any choice between these alternative estimates is inherently arbitrary, and leads to an arbitrary choice of QEP. Also, useful knowledge is ignored: if the estimator $s_1 s_{2,3}$ is used, then knowledge of the correlation between p_1 and p_2 is ignored, and similarly for the choice of $s_{1,2} s_3$. Ignoring correlations in this manner usually results in bad plan choices: the optimizer will be drawn toward those QEPs about which it knows the least, because use of the independence assumption makes these plans seem cheaper due to underestimation of the cardinality. For example, in the presence of a correlation between p_1 and p_2, the estimate $s_{1,2}$ is typically more accurate, but also much larger than the estimate $s_1 s_2$, which is based on an erroneous independence assumption. Even worse, if the optimizer does not use the same choice of estimate every time that it is required, then different QEPs will be costed inconsistently, leading to 'apples and oranges' comparisons and unreliable plan choices. Traditional RDBMSs have used cumbersome, ad hoc algorithms to enforce consistent choice of estimates [82, Appendix A].

Markl et al. [83] provide a method, called MAXENT, for consistent selectivity estimation that, similarly to ISOMER, is based on a maximum-entropy approach. Given a set of predicates $P = \{ p_1, p_2, \ldots, p_n \}$, the task is to determine the selectivity for each of the possible *atoms*—terms in disjunctive normal form—since then the selectivity for any predicate expressible as a Boolean formula in $p_1, p_2, \ldots, p_n$ can be computed as an appropriate sum of atom selectivities. For example, Fig. 16 displays the eight atoms corresponding to the case of $n = 3$ predicates; each atom corresponds to a basic region of the Venn diagram. The known selectivities, such as $s_{1,2}$ in Fig. 16, correspond in general to selectivities of unions of basic regions, and

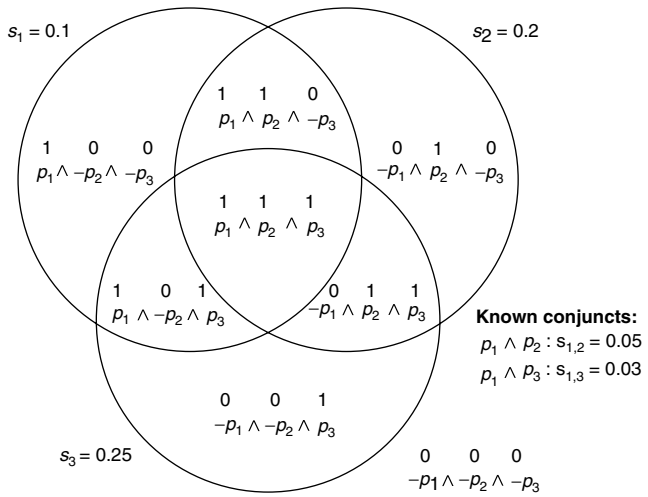

Fig. 16 Probability space for $N = \{1, 2, 3\}$ and $T = \{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \emptyset\}$.

hence to sums of atom selectivities. The maximum-entropy approach assigns atom-selectivity values in the 'simplest' manner that is consistent with the known selectivities.

In more detail, the maximum-entropy problem is formulated as follows. Denote each atom by a binary string of length n. For example, when $n = 3$, the string $b = 100$ denotes the atom $p_1 \wedge \neg p_2 \wedge \neg p_3$, and so forth; see, e.g. Fig. 16. Set $N = \{1, 2, \ldots, n\}$ and denote by 2^N the set of all subsets of N. For each predicate p_X with $X \in 2^N$, denote by $C(X)$ the set of *components* of X, i.e. the set of all atoms contributing to P_x. Formally, $C(\emptyset) = \{0, 1\}^n$ and

$$C(X) = \left\{ b \in \{0, 1\}^n \mid b_i = 1 \text{ for all } i \in X \right\}.$$

For example for predicates p_1 and $p_{1,2}$ we have

$$C(\{1\}) = \{100, 110, 101, 111\},$$

and

$$C(\{1, 2\}) = \{110, 111\}.$$

Also denoted by $T \subseteq 2^N$ the available *knowledge set*, i.e. the selectivities $\{ s_X \mid X \in T \}$ are assumed to be known and available to the optimizer. Given s_X for $X \in T$, MAXENT computes the selectivity s_X for $X \notin T$ by solving the following constrained optimization problem:

$$\underset{x_b | b \in \{0,1\}^n}{\text{minimize}} \sum_{b \in \{0,1\}^n} x_b \log x_b,$$

subject to the $|T|$ constraints

$$\sum_{b \in C(X)} x_b = s_X, \qquad X \in T,$$

where $x_b \in [0, 1]$ denotes the selectivity of atom b. The objective function is simply -1 times the entropy corresponding to an atom-selectivity assignment (hence we minimize rather than maximize). The constraints correspond to the known selectivities, and hence known sums of atom selectivities. By default, one of the included constraints is $s_\emptyset = \sum_{b \in \{0,1\}^n} x_b = 1$, which asserts that the combined selectivity of all atoms is 1. The solution is the atom-selectivity assignment having the maximum entropy value, subject to the constraints. Given this solution, we can compute any arbitrary selectivity s_X as $s_X = \sum_{b \in C(X)} x_b$. As with ISOMER, MAXENT uses iterative scaling to solve the optimization problem.

Figure 16 shows the probability space for the case $N = \{1, 2, 3\}$, $T = \{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \emptyset\}$, and selectivities $s_1 = 0.1$, $s_2 = 0.2$, $s_3 = 0.25$, $s_{1,2} = 0.05$, $s_{1,3} = 0.03$, and $s_\emptyset = 1$. We have the following six constraints:

(1.) $s_1 = x_{100} + x_{110} + x_{101} + x_{111} = 0.1$
(2.) $s_2 = x_{010} + x_{011} + x_{110} + x_{111} = 0.2$
(3.) $s_3 = x_{001} + x_{011} + x_{101} + x_{111} = 0.25$
(4.) $s_{1,2} = x_{110} + x_{111} = 0.05$
(5.) $s_{1,3} = x_{101} + x_{111} = 0.03$
(6.) $s_\emptyset = \sum_{b \in \{0,1\}^3} x_b = 1$

The task of selectivity estimation is to now compute a solution for all atoms $\{x_b \mid b \in \{0, 1\}^3\}$ that maximizes the entropy function $-\sum_{b \in \{0,1\}^3} x_b \log x_b$ and satisfies the above six constraints. Any desired selectivity s_X can then be computed from the x_b values as indicated previously. Figure 17 gives the results obtained when solving this constrained optimization problem. For instance, in this maximum-entropy solution, we obtain the selectivity estimates $s_{1,2,3} = x_{111} = 0.015$ and $s_{2,3} = x_{111} + x_{011} = 0.05167$.

A technical issue that must be addressed is the *a priori* elimination of atoms whose selectivity must be zero in any feasible solution of the optimization problem; see [83] for several solutions to this problem. MAXENT handles inconsistencies similarly to ISOMER by using linear programming and slack variables. A slight difference from ISOMER is that MAXENT uses positive slack variables not to eliminate a constraint, but rather to adjust the value of the right-hand side such that the modified constraint is satisfied. Thus the modified set of constraints represents a 'minimal' perturbation of the original constraint set such that there now exists at least one feasible solution.

Although the discussion so far has focused on sets of predicates such that all of the predicates in the set refer

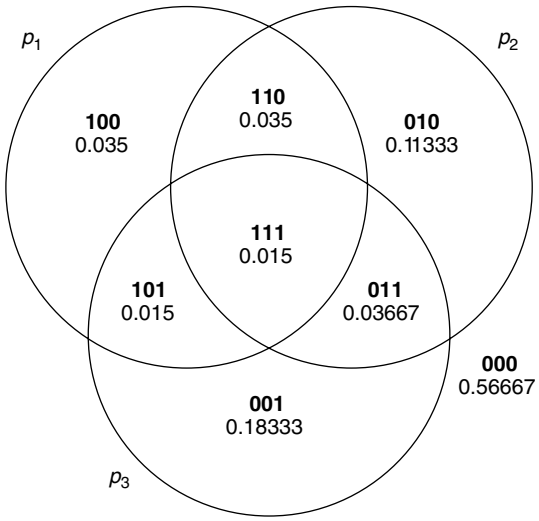

Fig. 17 Maximum-entropy solution for example problem.

to a single table, it is not hard to see that the MAXENT approach extends naturally to sets of predicates that refer to two or more tables, i.e. sets containing both local and join predicates. For the case of, e.g. $k = 2$ tables and in the absence of any other information, the maximum-entropy estimate of the selectivity of a join predicate is 0.5, since any pair of rows is as likely to join as to not join. In practice, enough information is usually available so that the estimated selectivity is much lower than 0.5. In one common scenario, for example, a join predicate p_i is known to be a key-to-foreign-key join, with table R_1 containing the key column and table R_2 containing the foreign-key column. Then the number of elements in the Cartesian product $R_1 \times R_2$ that satisfy p_i is $|R_2|$, because every value in the foreign-key column has, by definition, exactly one match in the key column. In this case, we add the element $X = \{i\}$ to the knowledge set T, with corresponding selectivity $s_X = 1/|R_1|$.

To handle large sets of predicates, MAXENT fixes a small integer constant μ and then (i) partitions the predicates, if possible, into mutually independent disjoint sets with at most μ predicates per partition, (ii) obtains a maximum-entropy solution for each partition, and then (iii) uses the independence assumption to combine the resulting partial selectivities via simple multiplication. (Independence of a given pair of predicates is indicated by the absence of a joint selectivity on these predicates in the knowledge set T.) If such a partitioning is impossible, then MAXENT forces a partitioning by removing nonsingleton elements from T until a partitioning can be found. See [83] for a greedy removal algorithm, in which the removed elements have the smallest deviations from the independence assumption, and hence the smallest impact on the quality of the maximum-entropy solution. Experiments on a prototype of the algorithm in

DB2 for LUW indicate that, using partitioning, MAXENT can produce high-quality estimates while adding only tens of milliseconds to the overall processing time, even for large numbers of predicates.

4.2. Prioritizing Maintenance

For both proactive and reactive approaches, maintaining the statistics on a table or view can consume considerable I/O and CPU resources, and there can be many tables on which to potentially collect or update statistics; the databases used by the SAP application, for example, typically contain tens of thousands of tables. The dedicated 'batch window' in the middle of the night, traditionally used to run such maintenance utilities, has pretty much disappeared because of globalization. How, then, can this necessary maintenance be performed without adversely affecting normal processing? Fortunately, the need to update statistics varies from table to table, depending upon how much the table has been updated since its statistics were collected last time. Often tables containing product lists, store locations, and organizational information are relatively static compared with tables containing transactions, inventory, or shipping status. It makes sense to give priority to the tables that have changed the most. Indeed, some sort of prioritization is essential, because there are virtually never enough system resources to process all of the tables in a database at the same time.

In DB2 for LUW, the RUNSTATS utility has been automated to run as a 'throttled' background task [84]. During certain 'maintenance windows,' RUNSTATS is allowed a larger share of CPU, and the statistics for the currently most 'important' tables are refreshed or collected. Without requiring user intervention, DB2 prioritizes tables by their 'urgency' with respect to statistics processing. The automated prioritization method [70] used by RUNSTATS first examines main-memory counters that record the number of updates, inserts, and deletes for each table. For those tables enjoying the most activity, RUNSTATS compares the current file size against the size as recorded in the system catalog and also samples the table to measure the degree to which the distribution of values in each column differs from the statistics last recorded for that column. Those frequently accessed tables having the greatest value changes are given highest priority for processing by RUNSTATS. This priority can also be influenced by query feedback, increasing

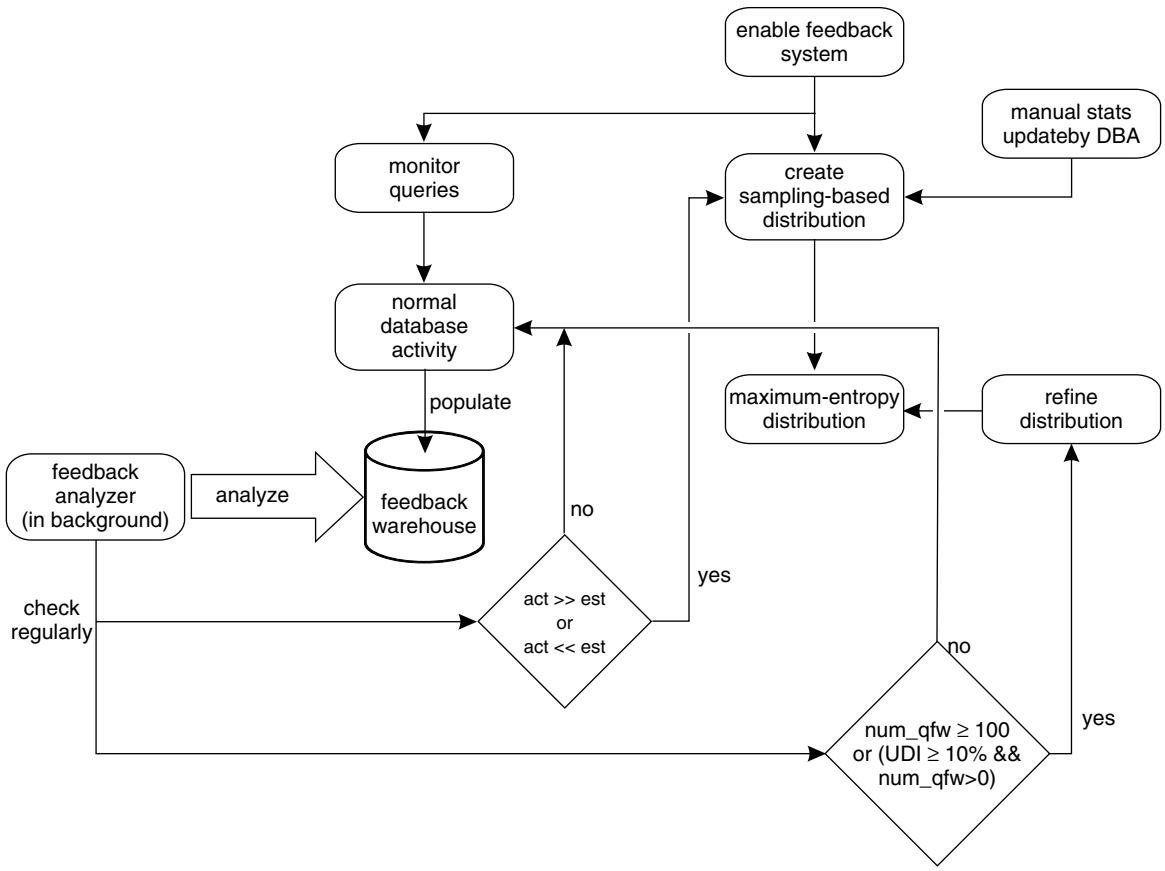

Fig. 18 Informix dynamic server architecture for automatically maintaining statistics.

the priority for those frequently accessed tables that have seen the largest deviations between the optimizer's selectivity estimates and the observed selectivities. Special care is taken to avoid 'starvation' scenarios in which statistics for a table are seriously out of date or have never been collected, yet RUNSTATS processing for the table is postponed indefinitely.

5. CONCLUSION

We have surveyed a variety of proactive and reactive techniques for discovering statistical features in relational data, primarily in the context of query optimization. There are many interesting and challenging problems still to be solved.

As indicated in Fig. 8, detection of pairwise correlations can go most of the way toward eliminating the worst selectivity-estimation errors. Nonetheless, discovering higher-order correlations can provide additional benefit. The question is how to detect and exploit such correlations in a practical manner. One possible approach is to extend relatively simple techniques such as CORDS or the reactive method of Section 3.1 to higher-order correlations; another approach is to improve the practical performance of techniques such as SASH.

One promising path for efficiently and robustly discovering statistical structure in relational (and other) data is to combine sampling-based proactive techniques with reactive techniques that exploit workload and/or query-feedback information. Some initial work in this direction includes the DB2 prioritization method described in Section 4.2, which combines query feedback and operation counting, and also uses proactive sampling to evaluate the adequacy of the current catalog statistics. The IDS prototype described in [80] also combines query feedback with sampling: an initial histogram on a column is created from a sample, and the histogram is then continually refined by applying the maximum-entropy method (as described at the end of Section 3.2). Whenever—as determined from query-feedback records—actual and estimated selectivities diverge to an unacceptably large degree, the histogram is reinitialized by taking a new sample, and the feedback-based refinement process then starts anew; see Fig. 18.

The proactive approaches described in this paper either use scanning, random sampling, or querying to learn about statistical properties of the data. The machine-learning and data mining communities, on the other hand, have been developing a theory of *active learning*—see, for example, Cohn et al. [85]—in which queries are generated in such a way as to maximally accelerate the learning process. Although most of this work has been focused on problems such as classification, it would be interesting to see if any of the ideas and techniques could be adapted to the database setting.

As we have seen, there has been much interest in dynamic approaches to query optimization, and hence to data gathering and discovery. This trend will likely continue, as database designers try to deal with new, complicated types of semistructured and unstructured data sources, as well as sophisticated queries over these sources. Another motivation behind this trend is the desire to simplify the DBMS architecture, especially the optimizer; see Bhattacharjee et al. [86] for a discussion of issues in next-generation database systems. Thus there will be an increasing emphasis on discovering statistical properties of data on the fly. Statistical technology for streaming data systems (e.g. Motwani et al. [87]) will be increasingly relevant.

Another emerging challenge to statistical discovery in databases is an increasing awareness that the data in a database is often inherently uncertain. Although there has always been a need to interpolate missing data, there has been a sharp increase in data uncertainty arising from entity-resolution processing in data integration, information extraction from unstructured text, measurement errors in sensor and radio frequency identification (RFID) data, and deliberate anonymization of data for privacy protection. As a result, database designers have been trying to develop 'probabilistic database systems' in which data uncertainty and lineage are integral concepts; see Dalvi and Suciu [88] for a recent survey, and also Jampani et al. [89] for a description of a Monte-Carlo-based system that can deal with real-world queries. A key challenge—over and above basic questions of how to formulate and answer queries on uncertain data—is how to handle the presence of this extra uncertainty when searching for statistical structure.

Overall, the problem of developing industrial-strength, efficient, scalable algorithms for discovering, and exploiting statistical structure in databases remains challenging, and provides many opportunities for research. We believe that the research efforts described in this paper can benefit from the varied skills that the database, statistics, and data mining communities can bring to bear; we hope to see fruitful collaborations in the future.

6. ACKNOWLEDGEMENT

The authors would like to thank the reviewers for their helpful comments, which resulted in an improved exposition.

REFERENCES

[1] M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, J. Gray, W. F. King III, B. G. Lindsay, R. A. Lorie, J. W. Mehl, T. G. Price, G. R. Putzolu, M. Schkolnick, P. G. Selinger, D. R. Slutz, H. R. Strong, P. Tiberio, I. L. Traiger, B. W. Wade, and R. A. Yost, System R: A relational data base management system, IEEE Comput 12(5) (1979), 42–48.

[2] E. F. Codd, A relational model of data for large shared data banks, Commun ACM 13(6) (1970), 377–387.
[3] G. Antoshenkov and M. Ziauddin, Query processing and optimization in Oracle Rdb, VLDB J 5(4) (1996), 229–237.
[4] P. Gassner, G. M. Lohman, K. B. Schiefer, and Y. Wang, Query optimization in the IBM DB2 family, IEEE Data Eng. Bull 16(4) (1993), 4–18.
[5] G. Graefe, The Cascades framework for query optimization, IEEE Data Eng Bull 18(3) (1995), 19–29.
[6] M. Jarke and J. Koch, Query optimization in database systems, ACM Comput Surv 16(2) (1984), 111–152.
[7] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price, Access path selection in a relational database management system, In Proceedings SIGMOD, Boston, MA, 1979, 23–34.
[8] Y. Matias, J. S. Vitter, and M. Wang, Wavelet-based histograms for selectivity estimation, In Proceedings SIGMOD, Seattle, WA, 1998, 448–459.
[9] R. Berinde, A. C. Gilbert, P. Indyk, H. Karloff, and M. J. Strauss, Combining geometry and combinatorics: a unified approach to sparse signal recovery, 2008, Preprint, Available at http://people.csail.mit.edu/indyk/rip2expand.pdf.
[10] A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss, Fast, small-space algorithms for approximate histogram maintenance. In Proceedings STOC, Montreal, Quebec, Canada, 2002, 389–398.
[11] J. Boulos, Y. Viemont, and K. Ono, A neural networks approach for query cost evaluation, Trans Inf Proc Soc Jpn 38(12) (1997), 2566–2575.
[12] H. Lu and R. Setiono, Effective query size estimation using neural networks, Appl Intell 16(3) (2002), 173–183.
[13] L. Getoor, B. Taskar, and D. Koller, Selectivity estimation using probabilistic models, In Proceedings SIGMOD, Santa Barbara, CA, 2001, 461–472.
[14] A. Balmin, T. Eliaz, J. Hornibrook, L. Lim, G. M. Lohman, D. E. Simmen, M. Wang, and C. Zhang, Cost-based optimization in DB2 XML, IBM Syst J 45(2) (2006), 299–320.
[15] J. Vitter, Random sampling with a reservoir, ACM Trans Math Softw 11(1) (1985), 37–57.
[16] V. Poosala, Y. E. Ioannidis, P. J. Haas, and E. J. Shekita, Improved histograms for selectivity estimation of range predicates, In Proceedings SIGMOD, Montreal, Canada, 1996, 294–305.
[17] M. Greenwald and S. Khanna, Space-efficient online computation of quantile summaries, In Proceedings SIGMOD, Santa Barbara, CA, 2001, 58–66.
[18] Q. Zhang and W. Wang, An efficient algorithm for approximate biased quantile computation in data streams, In Proceedings CIKM, Lisbon, Portugal, 2007, 1023–1026.
[19] Y. E. Ioannidis, The history of histograms (abridged), In Proceedings VLDB, Berlin, Germany, 2003, 19–30.
[20] M. Muralikrishna and D. J. DeWitt, Equi-depth histograms for estimating selectivity factors for multi-dimensional queries, In Proceedings SIGMOD, Chicago, IL, 1988, 28–36.
[21] T. Eavis and A. Lopez, rK-Hist: An R-tree based histogram for multi-dimensional selectivity estimation, In Proceedings CIKM, Lisbon, Portugal, 2007, 475–484.
[22] N. Thaper, S. Guha, P. Indyk, and N. Koudas, Dynamic multidimensional histograms, In Proceedings SIGMOD, Madison, WI, 2002, 428–439.
[23] H. Wang and K. C. Sevcik, A multi-dimensional histogram for selectivity estimation and fast approximate query answering, In Proceedings CASCON, Toronto, Ontario, Canada, 2003, 328–342.
[24] K. Beyer, P. J. Haas, B. Reinwald, Y. Sismanis, and R. Gemulla, On synopses for distinct-value estimation under multiset operations, In Proceedings SIGMOD, Beijing, China, 2007, 199–210.
[25] Y. Huhtala, J. Kärkkäinen, P. Porkka, and H. Tiovonen, TANE: An efficient algorithm for discovering functional and approximate dependencies, Comput J 42(2) (1999), 100–111.
[26] R. Agrawal and R. Srikant, Fast algorithms for mining association rules in large databases, In Proceedings VLDB, Santiago de Chile, Chile, 1994, 487–499.
[27] S. Bell and P. Brockhausen, Discovery of constraints and data dependencies in databases, Proceedings European Conference Machine Learning (ECML-95), Lecture Notes in Artificial Intelligence 914, Springer, Berlin, 1995, 267–270.
[28] J.-M. Petit, F. Toumani, J.-F. Boulicaut, and J. Kouloumdjian, Towards the reverse engineering of denormalized relational databases, In Proceedings ICDE, New Orleans, LA, 1996, 218–227.
[29] P. Godfrey, J. Gryz, and C. Zuzarte, Exploiting constraint-like data characterizations in query optimization, In Proceedings SIGMOD, Santa Barbara, CA, 2001, 582–592.
[30] J. Gryz, K. B. Schiefer, J. Zheng, and C. Zuzarte, Discovery and application of check constraints in DB2, In Proceedings ICDE, Heidelberg, Germany, 2001, 551–556.
[31] J. Edmonds, J. Gryz, D. Liang, and R. J. Miller, Mining for empty rectangles in large data sets, In Proceedings ICDT, London, UK, 2001, 174–188.
[32] J. Gryz and D. Liang, Holes in joins, J Intell Inf Syst, 26(3) (2006), 247–268.
[33] M. Siegel, E. Sciore, and S. Salveter, A method for automatic rule derivation to support semantic query optimization, ACM Trans Database Syst, 17(4) (1992), 563–600.
[34] C. T. Yu and W. Sun, Automatic knowledge acquisition and maintenance for semantic query optimization, IEEE Trans Knowl Data Eng 1(3) (1989), 362–375.
[35] R. Srikant and R. Agrawal, Mining quantitative association rules in large relational tables, In Proceedings SIGMOD, Montreal, Canada, 1996, 1–12.
[36] P. Brown, P. J. Haas, J. Myllymaki, H. Pirahesh, B. Reinwald, and Y. Sismanis, Toward automated large-scale information integration and discovery, Data Management in a Connected World, Springer, Berlin, 2005, 161–180.
[37] Y. Sismanis, P. J. Haas, and B. Reinwald, GORDIAN: Efficient and scalable discovery of all composite keys, In Proceedings VLDB, Seoul, Korea, 2006, 691–702.
[38] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh, Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals, J Data Min Knowl Discov 1 (1997), 29–53.
[39] D. Gunopulos, R. Khardon, H. Mannila, S. Saluja, H. Toivonen, and R. S. Sharma, Discovering all most specific sentences, ACM Trans Database Syst 28(2) (2003), 140–174.
[40] A. Deshpande, M. Garofalakis, and R. Rastogi, Independence is good: dependency-based histogram synopses for high-dimensional data, In Proceedings SIGMOD, Santa Barbara, CA, 2001, 199–210.
[41] L. Getoor and B. Taskar (eds), Introduction to Statistical Relational Learning, MIT Press, Cambridge, Massachusetts, 2007.
[42] S. Babu, P. Bizarro, and D. DeWitt, Proactive re-optimization, In Proceedings SIGMOD, Baltimore, Maryland, USA, 2005, 107–118.

[43] C. Chen and N. Roussopoulos, Adaptive selectivity estimation using query feedback, In Proceedings SIGMOD, Minneapolis, Minnesota, 1994, 161–172.
[44] B. S. Lee, L. Chen, J. Buzas, and V. Kannoth, Regression-based self-tuning modeling of smooth user-defined function costs for an object-relational database management system query optimizer, Comput J 47(6) (2004) 673–693.
[45] Q. Zhu and P. Larson, Building regression cost models for multidatabase systems, In Proceedings IEEE PDIS, Miami Beach, FL, 1996.
[46] Z. He, B. S. Lee, and R. R. Snapp, Self-tuning UDF cost modeling using the memory-limited quadtree, In Proceedings EDBT, Heraklion, Crete, Greece, 2004.
[47] N. Zhang, P. J. Haas, V. Josifovski, G. M. Lohman, and C. Zhang, Statistical learning techniques for costing XML queries, In Proceedings VLDB, Trondheim, Norway, 2005, 289–300.
[48] R. Gemulla, W. Lehner, and P. J. Haas, Maintaining Bernoulli samples over evolving multisets, In Proceedings PODS, Beijing, China, 2007, 93–102.
[49] R. Gemulla, W. Lehner, and P. J. Haas, Maintaining bounded-size sample synopses of evolving datasets, VLDB J 17(2) (2008), 173–202.
[50] P. J. Haas and C. König, A bi-level Bernoulli scheme for database sampling, In Proceedings SIGMOD, Paris, France, 2004, 275–286.
[51] P. G. Brown and P. J. Haas, Techniques for warehousing of sample data, In Proceedings ICDE, Atlanta, GA, 2006.
[52] M. Charikar, S. Chaudhuri, R. Motwani, and V. R. Narasayya, Towards estimation error guarantees for distinct values, In Proceedings PODS, Dallas, Texas, 2000, 268–279.
[53] P. J. Haas, Y. Liu, and L. Stokes, An estimator of the number of species from quadrat sampling, Biometrics 62(1) (2006), 135–141.
[54] P. J. Haas and L. Stokes, Estimating the number of classes in a finite population, J Am Stat Assoc 93(444) (1998), 1475–1487.
[55] E. Cohen, N. Grossaug, and H. Kaplan, Processing top-k queries from samples, In Proceedings 2nd Conference Future Networking Tech. (CoNext), Lisbon, Portugal, 2006, 7.
[56] M. Wu and C. Jermaine, A Bayesian method for guessing the extreme values in a data set, In Proceedings VLDB, Vienna, Austria, 2007, 471–482.
[57] C.-E. Särndal, B. Swensson, and J. Wretman, Model Assisted Survey Sampling, Springer, New York, 1992.
[58] P. G. Brown and P. J. Haas, BHUNT: Automatic discovery of fuzzy algebraic constraints in relational data, In Proceedings VLDB, Berlin, Germany, 2003, 668–679.
[59] H. Scheffé and J. W. Tukey, Nonparametric estimation. I. Validation of order statistics, Ann Math Stat, 16(2) (1945), 187–192.
[60] J. W. Tukey, Nonparametric estimation II. Statistically equivalent blocks and tolerance regions-The continuous case, Ann Math Statist 18 (1947), 529–539.
[61] I. F. Ilyas, V. Markl, P. J. Haas, P. G. Brown, and A. Aboulnaga, CORDS: Automatic discovery of correlations and soft functional dependencies, In Proceedings SIGMOD, Paris, France, 2004, 647–658.
[62] H. Cramér, Mathematical Methods of Statistics, Princeton University Press, Princeton, New Jersey, 1948.
[63] T. R. C. Read and N. A. C. Cressie, Goodness-of-Fit Statistics for Discrete Multivariate Data, Springer, New York, 1988.
[64] P. Bruni, T. Berman, J. Iczkovits, B. Soetarman, B. Steegmans, and M. Turner, DB2 9 for z/OS: New Tools for Query Optimization, IBM Redbook Series SG24-7421-00, IBM Corporation, Armonk, New York, 2007.
[65] S. Chaudhuri and V. Narasayya, Automating statistics management for query optimizers, IEEE Trans Knowl Data Eng 13(1) (2001), 7–20.
[66] A. El-Helw, I. F. Ilyas, W. Lau, V. Markl, and C. Zuzarte, Collecting and maintaining just-in-time statistics, In Proceedings ICDE, Istanbul, Turkey, 2007, 516–525.
[67] N. Bruno and S. Chaudhuri, Exploiting statistics on query expressions for optimization, In Proceedings SIGMOD, Madison, WI, 2002, 263–274.
[68] K. K. Chen, Influence query optimization with optimization profiles and statistical views in DB2 9: Optimal query performance in DB2 9 for Linux, UNIX, and Windows, 2006, Available at www.ibm.com/developerworks/db2/library/techarticle/dm-0612chen.
[69] M. Stillger, G. M. Lohman, V. Markl, and M. Kandil, LEO - DB2's LEarning Optimizer, In Proceedings VLDB, Rome, Italy, 2001, 19–28.
[70] A. Aboulnaga, P. J. Haas, M. Kandil, S. Lightstone, G. Lohman, V. Markl, I. Popivanov, and V. Raman, Automated statistics collection in DB2 UDB, In Proceedings VLDB, Toronto, Ontario, Canada, 2004.
[71] L. Lim, M. Wang, and J. S. Vitter, SASH: A self-adaptive histogram set for dynamically changing workloads, In Proceedings VLDB, Berlin, Germany, 2003, 369–380.
[72] V. Raman and G. Swart, How to wring a table dry: Entropy compression of relations and querying of compressed relations, In Proceedings VLDB, Seoul, Korea, 2006, 858–869.
[73] P. J. Haas, F. Hueske, and V. Markl, Detecting attribute dependencies from query feedback, In Proceedings VLDB, Vienna, Austria, 2007, 830–841.
[74] P. B. Gibbons, Y. Matias, and V. Poosala, Fast incremental maintenance of approximate histograms, In Proceedings VLDB, Athens, Greece, 1997.
[75] V. Poosala and Y. E. Ioannidis, Selectivity estimation without the attribute value independence assumption, In Proceedings VLDB, Athens, Greece, 1997, 486–495.
[76] A. Aboulnaga and S. Chaudhuri, Self-tuning histograms: Building histograms without looking at data, In Proceedings SIGMOD, Philadelphia, PA, 1999, 181–192.
[77] N. Bruno, S. Chaudhuri, and L. Gravano, STHoles: a multidimensional workload-aware histogram, In Proceedings SIGMOD, Santa Barbara, CA, 2001, 211–222.
[78] U. Srivastava, P. J. Haas, V. Markl, and N. Megiddo, ISOMER: Consistent histogram construction using query feedback, In Proceedings ICDE, Atlanta, GA, 2006.
[79] E. Amaldi and V. Kann, The complexity and approximability of finding maximum feasible subsystems of linear relations, Theor Comput Sci 147(1–2) (1995), 181–210.
[80] A. Behm, V. Markl, P. J. Haas, and K. Murthy, Integrating query-feedback based statistics into Informix Dynamic Server, In Proceedings BTW, Aachen, Germany, 2007, 582–601.
[81] A. Behm, DB2 learning optimizer, query feedback, frequent values and quantiles, Thesis Assistentenarbeit, Department of Information Technology, Berufsakademie Stuttgart, 2005.
[82] V. Markl, V. Raman, D. Simmen, G. Lohman, H. Pirahesh, and M. Cilimdzic, Robust query processing through progressive optimization, In Proceedings SIGMOD, Paris, France, 2004, 659–670.
[83] V. Markl, P. J. Haas, M. Kutsch, N. Megiddo, and T. M. Tran, Consistent selectivity estimation via maximum entropy, VLDB J 16(1) (2007), 55–76.

[84] S. S. Parekh, K. Rose, J. L. Hellerstein, S. Lightstone, M. Huras, and V. Chang, Managing the performance impact of administrative utilities, In Proceedings DSOM, Heidelberg, Germany, 2003, 130–142.

[85] D. A. Cohn, Z. Ghahramani, and M. I. Jordan, Active learning with statistical models, J Artif Intell Res 4 (1997), 129–145.

[86] B. Bhattacharjee, J. S. Glider, R. A. Golding, G. M. Lohman, V. Markl, H. Pirahesh, J. Rao, R. Rees, and G. Swart, Impliance: A next generation information management appliance, In Proceedings CIDR, Asilomar, CA, 2007, 351–362.

[87] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. S. Manku, C. Olston, J. Rosenstein, and R. Varma, Query processing, approximation, and resource management in a data stream management system, In Proceedings CIDR, Asilomar, CA, 2003.

[88] N. N. Dalvi and D. Suciu, Management of probabilistic data: foundations and challenges, In Proceedings PODS, Beijing, China, 2007, 1–12.

[89] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. M. Jermaine, and P. J. Haas, MCDB: a Monte Carlo approach to managing uncertain data, In Proceedings SIGMOD, Vancouver, BC, Canada, 2008, 687–700.