Histograms Reloaded: The Merits of Bucket Diversity

Carl-Christian Kanne
University of Mannheim
Germany
kanne@informatik.uni-mannheim.de

Guido Moerkotte
University of Mannheim
Germany
moerkotte@informatik.uni-mannheim.de

ABSTRACT

Virtually all histograms store for each bucket the number of distinct values it contains and their average frequency. In this paper, we question this paradigm. We start out by investigating the estimation precision of three commercial database systems which also follow the above paradigm. It turns out that huge errors are quite common. We then introduce new bucket types and investigate their accuracy when building optimal histograms with them. The results are ambiguous. There is no clear winner among the bucket types. At this point, we (1) switch to heterogeneous histograms, where different buckets of the same histogram possibly are of different types, and (2) design more bucket types. The nice consequence of introducing heterogeneous histograms is that we can guarantee decent upper error bounds while at the same time heterogeneous histograms require far less space than homogeneous histograms.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*; G.1.2 [**Numerical Analysis**]: Approximation

General Terms

Algorithms,Experimentation,Performance

1. INTRODUCTION

The plan generator is an essential part of every query compiler. It is responsible for generating plan alternatives, evaluating them, and choosing the best one. In the evaluation step, a cost model facilitates the assessment of plans. It consists of two vital parts. Cardinality estimation allows to estimate the size of intermediate results, and cost functions for algebraic operators calculate the final cost estimate.

Whereas cost functions for algebraic operators are very accurate (e.g. I/O cost estimations for different joins are typically less than three percent off the true execution times [4]) cardinality estimates still pose a real challenge.

SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.

Three decades after their first introduction to databases [9], histograms are still the prevailing means to provide cardinality estimates. A histogram partitions the (active) domain of an attribute into buckets. For each bucket, the number of distinct values it contains (d) and their cumulated frequency (f_c) are stored. This is true for virtually all histograms (see Sec. 4.1 on related work). Especially, this is true for those commercial DBMSs that we tested.

Sometimes, the average frequency ($\overline{f}$) instead of the cumulated frequency is recorded. However, since $\overline{f} = f_c/d$ this is equivalent. On the one hand, it is well known that the average minimizes the l_2 error, i.e., the sum of the squares of the differences between the true values and the estimates. On the other hand, the q-error, i.e. the factor by which the estimate deviates from the true value, is much more relevant for plan generation [11]. As a consequence, we challenge the paradigm to keep f_c and d.

After some preliminaries (Sec. 2), we start our investigation with an analysis of the q-error produced by three commercial systems. The results are depressing. Although we here concentrate on the simplest query kinds (exact match, distinct value, and range queries), the commercial systems produce estimates which are often several orders of magnitude off the true values (see Table 1).

The central goal of this paper is to find a way to produce histograms which exhibit a reasonable maximal q-error (say up to 2) while at the same time limiting space consumption to a reasonable amount. In a first step towards this goal, we design alternatives to the traditional bucket type, which stores f_c and d. We then construct q-optimal homogeneous histograms (see Sec. 4) for the traditional and the new bucket types. In homogeneous histograms, all buckets are of the same type, i.e. they store the same information. *Q-optimal histograms* obey a given upper bound for the maximal allowed q-error and consume the least space of all those who satisfy the given error bound. The results are mixed. One new bucket type will be superior to the traditional bucket type in many, but not all cases. Thus, there is no single clear winner for *all* datasets. Further, the space consumptions by all homogeneous histograms are subsatisfactory.

We then draw the only possible consequence and switch to heterogeneous histograms (Sec. 5). That is, histograms may now contain buckets of different types. Additionally, we add three more bucket types, which make sense only in the context of heterogeneous histograms. Two of them will use new results from approximation theory [11].

Heterogeneous histograms are not as simple to construct as homogeneous ones. Thus, we provide two detailed algorithms. One constructs q-optimal histograms but exhibits exorbitant runtimes. The other is a heuristics with moderate runtime but does not construct q-optimal histograms. The conclusion will be that the central goal as stated above can be met for almost all datasets.

Our whole investigation builds heavily on experiments with around 50 different real data sets. Only a small fraction of the results can be presented here. The results of all experiments are presented in a technical report.

2. PRELIMINARIES

Let R be a relation, A one of its attributes, and $\{x_1, \ldots, x_m\} = \Pi_A^D(R)$[1] the set of distinct values of A. Assume $i < j \Longrightarrow x_i < x_j$. We define the *spread* s_i as $s_i = x_{i+1} - x_i$ [13]. The *frequency density* is a set of pairs (x_i, f_i) with $f_i = |\sigma_{A=x_i}(R)|$, $1 \leq i \leq m$. The *cumulative frequency distribution* (CFD) is defined as a set of pairs (x_i, c_i) with $c_i = \sum_{j=1}^{i-1} f_i$.

2.1 Query Templates

We define three query templates for exact match queries (EMQ), distinct value queries (DCT) and range queries (RGE).

- $\mathtt{EMQ}(x) = |\sigma_{A=x}(R)|$
- $\mathtt{DCT}(ub, lb) = |\Pi_A^D(\sigma_{lb \leq A < ub}(R))|$
- $\mathtt{RGE}(ub, lb) = |\sigma_{lb \leq A < ub}(R)|$

The basic estimation task solved by using a histogram is to provide cardinality estimates for all three query templates for all possible parameters.

2.2 Q-Error

We now define the q-error following the definition of [11]. For alternative but equivalent definitions, see [1, 3]. Afterwards, we repeat some of the arguments used in the above papers to motivate the q-error.

For $z \in \mathbb{R}$, we define the quotient functional

$$||z||_Q = \begin{cases} \infty & \text{if } z \leq 0 \\ 1/z & \text{if } 0 < z \leq 1 \\ z & \text{if } 1 \leq z. \end{cases}$$

For $z > 0$, this is the same as saying $||z||_Q = \max(z, 1/z)$.

For a vector $z = (z_1, \ldots, z_m)^t \in R^m$, we define

$$||z||_Q = \max_{i=1}^{m} ||z_i||_Q. \tag{1}$$

We denote $|| \cdot ||_Q$ by l_q. However, be careful: l_q is *not* a norm. Subadditivity is the only one of the three properties required by a norm, which is satisfied by l_q.

Let $\vec{a}$ and $\vec{b}$ be two vectors in R^m, where $b_i > 0$. Define $\vec{a}/\vec{b} = \frac{\vec{a}}{\vec{b}} = (a_1/b_1, \ldots, a_n/b_n)^{\mathrm{T}}$. Then, we define the *q-error* of an estimation $\hat{b}$ of b as

$$||\hat{b}/b||_Q.$$

As l_∞, l_q produces valid, symmetric bounds for individual estimates. Define $q = ||\hat{b}/b||_Q$. Then,

$$(1/q)f_i \leq \hat{f}_i \leq qf_i.$$

[1] Π^D denotes projection with duplicate elimination.

Assume we have an estimate $\hat{f}_i$ for every f_i, $1 \leq i \leq m$, and $||\hat{f}_i/f_i||_Q \leq q$ for all i. Then

$$1/q \sum_{i=1}^{m} f_i \leq \sum_{i=1}^{m} \hat{f}_i \leq q \sum_{i=1}^{m} f_i. \tag{2}$$

Thus, adding estimates bounded by a q-error results in an estimate of the sum bounded by the same q-error. As usual, subtraction can be bad, as we will see below.

Besides these nice features, [11] presents several convincing arguments that the q-error is truly relevant in the context of query optimization. For the first time, they prove a connection between errors in cardinality estimates and plan costs. We present one important theorem. It gives an upper bound for the factor the optimal plan can be better than the plan produced by the plan generator under cardinality estimation errors.

THEOREM 2.1. *Let $C = C_{SMJ}$ or $C = C_{GHJ}$ be the cost function of the sort-merge or the Grace hash join. For a given query in n relations, let P be the optimal plan under the true cardinalities, $\hat{P}$ be the optimal plan under the estimated cardinalities, $C(P)$ be the true costs under C of the optimal plan, and $C(\hat{P})$ be the true costs under C of the plan produced under the estimated cardinalities. Then*

$$C(\hat{P}) \leq q^4 C(P),$$

where q is defined as

$$q = \max_{x \subseteq X} ||\hat{s}_x / s_x||_Q,$$

with X being the set of relations to be joined, and s_x ($\hat{s}_x$) is the true (estimated) size of the join of the relations in x. That is, q is the maximum estimation error taken over all intermediate results.

This bound is rather tight, as shown in [11].

Although the q-error should be the error metrics of choice, it is rarely used in the literature. The exceptions are [1, 3, 11]. These papers provide more good arguments for using the q-error. Thus, it is our error metrics of choice.

3. THE COMMERCIAL STATE OF THE ART

In this section, we take a look at the estimation quality of three commercial database systems.

3.1 Data Sets

For our experiments, we decided to use real data instead of generated data, since we feel that generated data tends to be easier to approximate than real data. Further, for real data one cannot argue that they are unrealistic.

Let us briefly describe the datasets used and their provenance. By `citeseer` (cs), we denote a 2006 instance of the citeseer database of the 10.000 most cited computer science authors. The number of citations is the only attribute we consider here. By `ecb_usdeur` (ecb), we denote the exchange rates between Euro and US Dollar as fixed daily by the Eureopean Central Bank (www.ecb.de). By `uniprot`, we denote the swiss protein database (www.uniprot.org). Here, we consider two numerical attributes: the protein length measured in the number of amino acids (`uniprotAA`, AA) and the protein's molecular weight (`uniprotMW`, MW). By `weather`, we denote the database of weather measurements of all stations over the world provided by the World Meteorological Organization (www.ncdc.naoa.gov). It provides measurements from 1929-2009. From this relation, we take the attributes *sea level pressure* (`wtrslp`, slp) and *temperature* (`wtrtmp`, tmp).

We experimented with about 50 datasets which exhibit different degrees of difficulty difficulty to approximate them. The samples presented in this paper were choosen because they cover this range. However, the chosen datasets are not representative, but are biased towards the difficult end. The results for all datasets are represented in a technical report [10].

3.2 Database Management Systems

We loaded the datasets into three commercial systems. We named them System X, System Y and System Z. We ran the statistics collection without sampling. That is, we told the systems to do a full scan, since we wanted to avoid the additional hazard often introduced by sampling [1]. All three systems use (different kinds of) histograms. However, they have one thing in common: for each bucket, they store the number of distinct values and their cumulative frequencies. For details, see Sec. 4.1. Two of the three systems have a comparable built-in upper limit for the number of buckets in their histograms. The third database system provides a tuning knob for it. We chose the parameter such that it equals the one of the others having the higher upper limit for the number of buckets.

The maximal space consumption for one of the commercial systems could be easily determined. It is around 3,200 Bytes. We will use this number subsequently to formulate some of the questions and to discuss some of the results.

3.3 Queries and Estimates

For all data sets, we generated all possible exact match queries (EMQ) and asked the commercial database systems to estimate their result cardinalities. This was done via their explain utitilities. For each dataset, we systematically generated all possible ranges by pairing all `lb` $<$ `ub` pairs taken from the attribute's active domain. For large domains, this results in too many queries, since the slowest of the commercial database systems could process only 5-7 queries per second. In these cases, we stopped the process if a couple of million queries were generated. Thereby, we left out larger ranges, since these are typically easier to approximate than smaller ones. Even though we limited the number of queries, the slowest system needed a couple of weeks to process them.

For every range, we generated two queries: one returning the distinct values within the range (DCT) and the other returning the tuples with an attribute value within the range (RGE).

3.4 Results

We ran all queries and compared the estimate calculated by a commercial system with the true value. In order to do so, we calculated the q-error of every query. Then, for a given data set, we counted the number of queries of a certain kind for which a given system produced a q-error less than or equal to 2, 3, 4, 5, or larger than 5. Additionally, we calculated the maximal q-error occuring for a given dataset, query kind, and commercial system. The results are shown in Table 1.

First, we note that System X provides relatively good estimates on the first two datasets for all estimation tasks. Still, a worst case error of 7 or 11 is unbearable. Remember that the q-error is multiplicative. Hence, between the true cardinality and estimated cardinality lies a factor of 11 for the worst estimate provided by System X for range queries on citeseer. We observe still higher errors like 66, 282, 4446, and 108488. Remember that the factor between the costs of the plan generated due to cardinality estimation errors and the optimal plan can be bound by the q-error taken to the power of 4. This bound is relatively tight. Taking 5 to the power of 4 already yields 625, for 10 we get 10,000.

Q-errors of this magnitude essentially make plan generation a random process. This might be the reason vendors strive to incorporate a syntax for plan hints into their SQL parser. However, we cannot see how this helps. The chances are that the user makes equal errors in estimating the cardinalities and, hence, provides the wrong plan hints.

3.5 Goal

Now, we can state the goal of the paper. After seeing these subsatisfactory results, we asked ourselves whether it would be possible to limit the maximal q-error for all EMQ, DCT, and RGE queries to 2 and at the same time restrict the histogram size to 3,200 Bytes. We used this number as commercial vendors are clearly willing to spend that much memory for statistics on single attributes. The answer will be that for some datasets we can, for others we cannot (yet?). We thus relax the memory constrained after some cost calculations in Sec. 4.4.

Given a maximal q-error of 2, we call a dataset *simple* if this goal can be achieved. We call it *tractable* if its histogram needs less than 10KB of memory and *challenging* otherwise.

4. HOMOGENEOUS HISTOGRAMS

4.1 Related Work

There are many kinds of histograms, for example, variable-range histograms (equi-depth) [9, 8], variable-count (equi-width) histograms [9, 8], variable-range & variable-count histograms [9], quantile-based histograms [12], serial histograms [5, 6], v-optimal histograms [7], end-biased histograms [6], maxdiff histograms [13], and so on. They all have one thing in common: for every bucket they store the number of distinct values occurring in the bucket and their average frequency.

There is one notable exception to this rule. König and Weikum proposed to approximate the frequencies in a bucket by a polynomial of degree one derived by linear regression. However, this approach cannot guarantee a low q-error [11], which is not surprising because linear regression minimizes l_2. The same is true for all the other histogram types.

The histogram implementations of commercial systems also store per bucket the number of distinct values occurring in it and their average (or, equivalently, cumulative) frequency. However, one commercial system implements a very appealing idea: it also stores the precise frequencies of the bucket boundaries. This appears to be attractive in the presence of outliers if they happen to coincide with a bucket boundary.

The core of Moerkotte et al. [11] consists of a method to find best approximations with minimal q-error. However, their paper is very weak on experiments, and does not discuss how to construct full-fledged histograms. We reme-

	System X			System Y			System Z		
	EMQ	DCT	RGE	EMQ	DCT	RGE	EMQ	DCT	RGE
cs									
≤ 2	1818	829883	829716	523	262121	829942	894	259367	829657
≤ 3	86	705	821	379	176602	575	898	177422	638
≤ 4	10	134	165	70	160427	183	107	162303	175
≤ 5	1	54	73	863	189378	74	17	189856	85
> 5	1	59	60	81	42307	61	0	41887	280
max	6	11	11	9.2	27	12	4.2	166667	166667
ecb									
≤ 2	2031	571406	571187	408	5184	568664	1989	564369	565174
≤ 3	30	498	647	0	235209	1176	65	2270	1393
≤ 4	3	73	120	1635	322792	274	8	377	567
≤ 5	0	9	26	1	5068	121	2	418	227
> 5	0	1	7	20	1999	17	0	4553	4626
max	4	6	7	20	18	8.3	4.6	1449	3009
AA									
≤ 2	2301	1078271	1079142	364	155603	1016976	628	158638	1015302
≤ 3	154	21974	20074	182	106318	65084	218	108486	59132
≤ 4	35	25137	25856	196	89312	28379	133	90961	21648
≤ 5	18	5207	4777	238	122671	9690	138	122722	10891
> 5	14	2640	3380	1542	659325	13100	1405	652422	26256
max	18	41.5	52	62	282	66	33.6	3451	24159
MW									
≤ 2	64700	4199819	4178481	67009	3872563	4213096	34254	3824543	4200265
≤ 3	6642	35406	52547	4602	337455	41805	37612	377724	50356
≤ 4	906	7807	11003	772	33510	7856	653	38917	9526
≤ 5	203	5066	5632	121	9195	2846	0	9728	3319
> 5	68	28694	29129	15	24069	11189	0	25880	13326
max	13	102.5	102.5	8.5	39	71.5	3.8	3448	5172
slp									
≤ 2	784	771454	478694	158	93222	452840	232	102426	453939
≤ 3	101	0	43469	91	86278	33607	94	93432	32403
≤ 4	88	0	26517	76	64070	23726	59	70201	22629
≤ 5	62	0	17590	60	35725	16545	47	41115	15706
> 5	762	0	205184	1412	492159	244736	1365	464280	246777
max	1046	2	1383	46349	346	108488	8074	342	695929
tmp									
≤ 2	1443	990831	769040	309	174292	769182	429	173920	773043
≤ 3	119	137	43524	293	158315	44080	176	158163	41818
≤ 4	86	26	26755	393	203644	27104	153	202554	26412
≤ 5	75	11	21253	111	26210	21514	203	27643	21547
> 5	514	9	130442	1131	428553	129134	1276	428734	128194
max	543	8	4446	49458	145	24729	31524	102	14928

Table 1: Errors produced by commercial histograms

dy these deficiencies. In particular, they do not treat DCT queries. For RGE queries, they propose to use an approximation of the cumulated frequency distribution (CFD). The latter approach undermines our goal to minimize the q-error, as we next illustrate with a simple example.

Assume for some $i < j$ and the cumulative frequency distribution (x_i, c_i), we have $c_i = 540$ and $c_j = 660$. Assume the maximal q-error of the approximation is 1.1 and the estimates for c_i and c_j are $\hat{c_i} = 594$ and $\hat{c_j} = 600$. Then, using the difference of the CFD results in a true value for the cardinality of RGE(x_i,x_j) = 120. The difference of the estimates gives 6. Thus, the q-error becomes 20, which is far worse than the 1.1 we started with. Hence, using CFD (as is done in [11]) is no option.

4.2 Bucket Types

We call a bucket containing the number of distinct values and their cumulated frequency (or, equivalently, average) a *traditional bucket* and denote its type by `T-`. If it additionally stores the boundary frequency of its lower boundary, we denote its type by `T-B`.

Assume we have a set of frequencies F for the distinct values in a bucket. How can we minimize the q-error if we have to approximate F by a single number? The answer is simple, we only have to return the q-middle of F, which is defined as $\sqrt{\min(F)\max(F)}$ [11]. To see this, remember that the q-error is multiplicative in nature. Thus, instead of storing the cumulated frequency, we can store the q-middle of the frequencies. This gives bucket type `-Q-`. Adding the boundary frequencies yields bucket type `-QB`.

For an exact match query or small range queries, the usage of the q-middle is favorable. However, with larger ranges, the q-error produced by the average converges against the minimal value of 1. Thus, it might make sense to store both the cumulated frequency and the q-middle of the frequencies. In this case, one must add a threshold for a range width, indicating which approach to use for cardinality estimation. This results in bucket type `TQ-`. Adding the boundary frequency yields bucket type `TQB`.

Additionally, all the above bucket types store the number of distinct values they contain.

4.3 Histogram Construction

For a given bucket type and a maximal allowed q-error q, we say a histogram is *q-optimal* if the following conditions hold:

1. For every EMQ, DCT, RGE query the maximal q-error produced by the histogram is at most q.
2. Among all histograms satisfying condition one, it uses the smallest number of buckets.

Note that since in a homogeneous histogram all buckets have the same size, minimizing the number of buckets is equivalent to minimizing space usage.

We now sketch an efficient algorithm to construct q-optimal homogeneous histograms. The idea of this algorithm was already presented in [11]. The main observation that led to this idea is that the q-error produced by a bucket for the various estimation tasks increases monotonically with the number of distinct values it contains. The general algorithm starts with the smallest distinct value of the attribute's active domain and then finds the largest number of consecutive distinct values which still meet the given error bound. The smallest unprocessed domain value then becomes the start of the next bucket. The process proceeds until all domain values are covered. The largest bucket which still satisfies the given q-error bound can be found by a binary search.

4.4 Evaluation

For all four bucket types, we generated the q-optimal homogeneous histogram for all datasets. According to our goal, we used an upper bound of 2 for the maximal allowed q-error. The results are summarized in Table 2. It contains the number of bytes consumed by the q-optimal homogeneous histograms constructed for a certain bucket type and dataset.

We can make the following observations:

1. The bucket type `-Q-` is superior to `T-` in all cases but one.
2. There is no way to meet the 3,200 Byte boundary while at the same time requiring a maximum q-error of 2.
3. No single bucket type used in a q-optimal homogeneous histogram is superior in all cases.

The last observation has motivated us to consider histograms which may contain different bucket types at the same time. We call these *heterogeneous histograms*. The resulting sizes are shown in column HetHeu. They are the topic of the next section.

Note that the histograms mentioned in Sec. 4.1 on related work, which store for each bucket the number of distinct values and their cumulated (average) frequency, cannot be better than the q-optimal homogeneous histograms with bucket type `T-`.

5. HETEROGENEOUS HISTOGRAMS

Heterogeneous histograms possibly contain buckets of different types. This, of course, requires a bucket descriptor, which holds the bucket's type. Some buckets need additional parameters. As a consequence, bucket descriptors are 1-2 bytes long.

The bucket types described in the previous section can all be contained in a heterogeneous histogram. However, they do not suffice. Hence, we introduce three more bucket types in Sec. 5.1. Sec. 5.2 discusses two algorithms to construct heterogeneous histograms. The first one produces q-optimal heterogeneous histograms but is prohibitively expensive, the other one is a heuristics. Sec. 5.3 presents the experiment findings.

5.1 Bucket Types

We use the verb " to approximate" in the context of a set of pairs $S = \{(x_i, y_i)\}$ of numbers to denote the construction of the best approximation of S under l_q by either a polynomial or a function e^p for some polynomial p, whichever is better. To derive the best approximation, we use the algorithm presented in Appendix A, which is faster and easier to understand than the one from [11].

The problem the first two bucket types will solve is the following. Any estimation for queries of type DCT or RGE essentially have two input parameters: the lower bound of the range and the upper bound of the range. Since the approximation methods developed in [11] only provide one-

dataset	T-	T-B	-Q-	-QB	TQ-	TQB	HetHeu
citeseer	6318	7565	5143	5698	8736	9875	1125
ecb_usdeur	9217	12818	10127	11411	14952	18100	7639
uniprotAA	5837	6698	4735	5137	8127	8925	952
uniprotMW	289809	329477	226746	251048	393120	428700	39277
wtrslp	6929	4080	6578	8470	10395	12425	4039
wtrtmp	6708	5695	6617	8137	10605	11975	5210

Table 2: Sizes of q-optimal homogeneous histograms

dimensional approximations, we need to reduce the two-dimensional problem to a one-dimensional one. Each new bucket type builds on one reduction method.

There are two simplifications we apply to all (old and new) bucket types.[2] If the domain is integer and the bucket contains all possible integers, we call it *dense*. For dense buckets, we do not need to take any measures for RGE and DCT, since they can be derived from the EMQ information. If in a bucket all frequencies equal one, we do not need to take any measures for EMQ and RGE, since these are derivable from the DCT information. To capture these situations, we reserve two bits in the bucket descriptor.

5.1.1 Width-Based Approximation

We introduce width-based approximation buckets. Every bucket of this type contains at most three approximations: one for each of EMQ, DCT, RGE. To handle the EMQ case, the frequency density is approximated. Assume the bucket to be build spans the interval $[x_i, x_j[$.

The general idea of this kind of bucket is to approximate for a range $[\mathrm{lb}, \mathrm{ub}[$ contained in the bucket the number of distinct values and their cumulative frequency by a function in the range query's width $\mathrm{ub} - \mathrm{lb}$. This approximation can be constructed as follows. Imagine a sliding window of width w, which moves over all $x \in [x_i, x_j[$ which additionally satisfy $x + w \leq x_j$. Fix w and let x take every possible value. Then, we can take the minimum and maximum value over all x of the number of distinct values and their cumulated frequency. For every window width w, we can therefore calculate the q-middle of the number of distinct values (d_w) occurring in a range of width w and the cumulated frequency of these distinct values (c_w). Since d_w and c_w only depend on w, we can build functions $\hat{f}_{\mathrm{DCT}} : w \to d_w$ and $\hat{f}_{\mathrm{RGE}} : w \to c_w$. Of course, considering all possible x is unfeasible for non-integer domains. Therefore, we use only those x_i which occur in a bucket.

Let $[x_i, x_j[$ be the interval for which we want to construct a bucket. Define the set of widths

$$W = \{x_l - x_k | i \leq k < l \leq j\}.$$

Then, width-based approximation buckets hold approximations of the following two sets for DCT and RGE:

$$\{(w_k, \text{q-middle}(S_{\mathrm{DCT}})) | 1 \leq k \leq n\}$$

and

$$\{(w_k, \text{q-middle}(S_{\mathrm{RGE}})) | 1 \leq k \leq n\}$$

[2]These simplifications were also applied during the construction of q-optimal homogeneous histograms.

where

$$\begin{aligned} S_{\mathrm{DCT}} &= \{\mathrm{DCT}(x_l, x_l + w_k) | i \leq l < j, x_l + w_k < x_j\} \\ S_{\mathrm{RGE}} &= \{\mathrm{RGE}(x_l, x_l + w_k) | i \leq l < j, x_l + w_k < x_j\}. \end{aligned}$$

Let us illustrate this definition by a simple example. Consider the frequency distribution

$$(1,1), (2,2), (3,3), (4,4), (5,5)$$

and the bucket $[1, 5[$. We see that $W = \{1, 2, 3, 4\}$. The following table contains for every $w_k \in W$ the set $\mathrm{DCT}(x_l, x_l + w_k)$ and the q-middle:

w_k	$\mathrm{DCT}(x_l, x_l + w_k)$	q-middle
1	$\{1, 2, 3, 4\}$	2
2	$\{3, 5, 7\}$	$\sqrt{21}$
3	$\{6, 9\}$	$\sqrt{54}$
4	$\{10\}$	10

The last step is to find the best approximation under l_q for the set

$$\{(1, 2), (2, \sqrt{21}), (3, \sqrt{54}), (4, 10)\},$$

using the methods presented in [11].

5.1.2 Bucklet-Based Buckets

Every bucket of this type contains at most three approximations: one for each of EMQ, DCT, RGE. To handle the EMQ case, the frequency density is approximated. Assume the bucket to be built spans the interval $[x_i, x_j[$.

For a fixed width w, imagine a window which starts at position $x \in [x_i, x_j[$ and for which $x + w <= x_j$ holds. We call such a window a bucklet. For each bucklet $[x, x + w[$, we calculate the number of distinct values d_x and their cumulated frequency c_x. Since w is fixed, these numbers only depend on the position x. The idea of bucklet-based buckets is to approximate the set of pairs (x, d_x) and (x, c_x) for some positions x.

We define a series of bucklets of width w as $[x_l, x_l + w[$ such that $i \leq l \leq j$ and $x_l + w \leq x_j$. For each bucklet, we calculate $\mathrm{DCT}(x_l, x_l + w)$ and $\mathrm{RGE}(x_l, x_l + w)$. Then, we approximate the sets $\{(x_l, \mathrm{DCT}(x_l, x_l + w))\}$ and $\{(x_l, \mathrm{RGE}(x_l, x_l + w))\}$. This results in two approximation functions $\hat{f}_{\mathrm{DCT}}$ and $\hat{f}_{\mathrm{RGE}}$. These functions can then be used to approximate the cardinality of a DCT/RGE query. For a given range $[\mathrm{lb}, \mathrm{ub}[$, we sum up the terms

$$\hat{f}_{\mathrm{DCT/RGE}}(a_l) * w / (b_l - a_l)$$

for all intersections $[a_l, b_l[$ of $[\mathrm{lb}, \mathrm{ub}[$ with some bucklet $[x_l, x_l + w[$.

Let us illustrate the construction by a simple example. Consider again the frequency distribution

$$(1,1), (2,2), (3,3), (4,4), (5,5)$$

```
BestBucket(i,j,q,T)
  Input:  indices i,j,
        maximal q-error q,
        set of bucket types T
  Output:  bucket or 0
  r = 0; // will be result bucket or 0
  for all t ∈ T
    b = bucket of type t for [x_i, x_j[
    if (b.qerror <= q and // for all of EMQ, DCT, RGE
      (0 == result or b.size < r.size))
      r = b;
  return r;
```

Figure 1: Procedure BestBucket

and the bucket $[1, 5[$. For a bucket width $w = 2$, we see that the (x, d_x) pairs are

$$(1, 3), (2, 5), (3, 7).$$

For this set, we calculate the best approximation $\hat{f}_{\mathrm{DCT}}$ under l_q, using the methods presented in [11].

For our experiments, we have used five times the minimal spread found in the bucket as w.

5.1.3 Q-Compression Buckets

In many datasets, there are parts which are not approximable such that the given error bound is met. Assume the maximal allowed q-error is 2 and we limit the degree of the approximating polynomial to 2. Consider the three datapoints $(1, 1)$, $(2, 18)$, and $(3, 3)$, which we would like to approximate for the EMQ case. The best linear approximation under l_q for these data points is $f(x) = 3x$, which results in a q-error of 3.

For unapproximable parts, we could use *exact buckets*, which store the precise frequency density (x_i, f_i) for all x_i falling into the bucket. However, we can do better. For a given maximal allowed q-error q, any number in the interval $[q^{2l}, q^{2(l+1)}]$ can be approximated by q^{2l+1}, since

$$||q^{2l+1}/x||_Q \leq q$$

for all $x \in [q^{2l}, q^{2(l+1)}]$. Thus, if $f_{\max}$ is the maximal occurring frequency, and $f_{\max} \leq q^{2(k+1)}$ for some k, then $\lceil \log_2(k) \rceil$ bits suffice to encode the frequency. Q-compression buckets do exactly this.

5.2 Histogram Construction

Before we discuss the two algorithms to construct heterogeneous histograms, we introduce the simple procedure `BestBucket` (see Fig. 1) to construct the best bucket for a given range $[x_i, x_j[$. It is parameterized with the maximal allowed q-error (q) and a set of bucket types (T). The latter is necessary since T will differ in the two construction algorithms. Note that it returns 0 if no bucket type in T can meet the given error bound. Meeting the error bound means that for all queries of all types EMQ, DCT, RGE the q-error is less than or equal to q.

5.2.1 Optimal Heterogeneous Histograms

The algorithm to construct optimal heterogeneous histograms (see Fig. 2) implements a simple memoization strategy [2]. The map `BestHistogram` holds for every bucket

```
HetOpt(i, j, q, T_opt)
Input:  indices of histogram boundaries i, j
        maximal q-error q,
        set of bucket types T_opt
Output:  optimal heterogeneous histogram
  if (0 == BestHistogram(i,j))
    BestHistogram(i,j) = BestBucket(i,j, q, T_opt);
    for (k = i+1; k ≤ j; ++k)
      left = HetOpt(i,k, q, T_opt);
      right = HetOpt(k,j, q, T_opt);
      if (left.size + right.size <
          BestHistogram(i,j).size)
        BestHistogram(i,j) = left ∘ right;
  return BestHistogram(i,j);
```

Figure 2: Construction of optimal heterogenous histograms

$[x_i, x_j[$ the best (smallest in size) histogram under the given maximal q-error q. It is filled recursively by considering (1) a single bucket over the whole range and then (2) all possible splits of the range into two parts. For each part (`left`, `right`) the optimal heterogeneous histogram is constructed recursively. The best alternative is kept by concatenating the histograms. (We assume that a histogram is a sequence of buckets and denote the concatenation of histograms by '$\circ$'.) The set of bucket types $\mathrm{T}_{\mathrm{opt}}$ contains all bucket types except exact buckets. Since q-compression buckets always meet a given error bound, no `BestBucket` call in `HetOpt` ever returns 0. The top-level call is `HetOpt`(1,m,q,$\mathrm{T}_{\mathrm{opt}}$), where m is the number of pairs of the frequency density. The performance of `HetOpt` will be discussed below.

5.2.2 Heuristics

The heuristics `HetHeu` to construct heterogeneous histograms uses the subroutine `FindLargest` (Fig. 3). Its main idea is the same as the one of the procedure presented in Sec. 4.3. For a given lower bucket boundary x_i, it tries to construct the largest bucket that still meets the given q-error bound q. However, we must be careful, since exact buckets and q-compression buckets always meet the error bound but grow in size with the number of distinct values they contain. Thus, the set of bucket types $\mathrm{T}_{\mathrm{heu}}$ considered by `FindLargest` excludes these two bucket types.

`HetHeu` iteratively calls `FindLargest` to find the bucket of a type in $\mathrm{T}_{\mathrm{heu}}$ which still meets the given error bound and consumes the least space among them. This bucket is then appended to the histogram. Its upper bound becomes the lower bound of the bucket to be constructed next.

For some badly approximable buckets, it might be beneficial to introduce q-compression buckets. Whenever this leads to space savings, we replace sequences of buckets in the histogram constructed so far by a q-compression bucket. This is done by `Compactify`, which is called at the end of `HetHeu`. Its implementation is quite simple. It systematically looks for subsequences in the histogram, which can beneficially be replaced by a q-compression bucket.

5.3 Evaluation

The maximal degree of the polynomials used for approximations in width-based and bucklet-based buckets is a pa-

citeseer				
q-err	#bkts	size	bits	cpu
1.7	15	1373	6	11.337
2	27	1125	5	15.797
2.3	29	901	4	22.533
2.5	25	824	3	26.862
2.7	25	746	3	42.775
3	26	635	3	47.375
3.3	21	546	2	57.808
3.5	21	489	2	74.129
3.7	21	471	2	80.049
4	25	420	2	83.233
4.5	19	314	1	82.101
5	14	220	1	112.431

ecb_usdeur				
q-err	#bkts	size	bits	cpu
1.7	402	8558	33	12.5
2	467	7639	30	14.6
2.3	421	6558	25	13.0
2.5	348	5636	22	11.3
2.7	299	4861	19	10.5
3	224	3867	15	10.0
3.3	176	3156	12	10.5
3.5	147	2678	10	12.0
3.7	128	2336	9	15.2
4	95	1820	7	20.8
4.5	63	1242	5	36.6
5	49	919	4	58.8

uniprotAA				
q-err	#bkts	size	bits	cpu
1.7	26	1261	4	18.9
2	28	952	3	33.9
2.3	25	917	3	49.8
2.5	32	791	3	53.8
2.7	33	702	2	67.1
3	29	626	2	67.8
3.3	24	555	2	74.5
3.5	23	521	2	78.2
3.7	20	476	2	78.8
4	19	450	1	75.0
4.5	18	392	1	104.2
5	21	376	1	106.7

uniprotMW				
q-err	#bkts	size	bits	cpu
1.7	270	50504	6	346
2	692	39277	4	309
2.3	805	33984	4	371
2.5	974	29585	3	652
2.7	821	23826	3	1456
3	689	18996	2	2167
3.3	649	16893	2	2648
3.5	625	15883	2	2802
3.7	596	14657	2	3757
4	636	13475	1	3951
4.5	578	10673	1	4802
5	466	8118	1	6674

wtrslp				
q-err	#bkts	size	bits	cpu
1.7	370	5838	26	4.6
2	227	4039	18	5.4
2.3	130	2041	9	11.6
2.5	13	255	1	68.7
2.7	11	220	1	66.0
3	10	194	1	59.0
3.3	41	574	3	30.9
3.5	8	149	1	115.8
3.7	32	459	2	40.4
4	27	367	2	43.4
4.5	21	288	1	56.8
5	18	241	1	84.7

wtrtmp				
q-err	#bkts	size	bits	cpu
1.7	457	6949	25	5.2
2	357	5210	19	5.9
2.3	136	2390	9	19.5
2.5	87	1515	5	36.6
2.7	69	1180	4	57.4
3	51	830	3	94.2
3.3	40	609	2	106.6
3.5	34	490	2	145.8
3.7	31	456	2	167.1
4	26	381	1	168.2
4.5	21	308	1	188.6
5	18	264	1	189.9

Table 3: Detailed results for heterogeneous profiles (Sizes in bytes, CPU in seconds)

```
FindLargest(i, q, T_heu)
Input:  index i of lower bucket boundary
        maximal q-error q,
        set of bucket types T_heu
Output:  highest index j for upper boundary
  by binary search
    find largest j > i such that
      b = BestBucket(i,j, q, T_heu) succeeds
    return j, b;

HetHeu({x_i, f_i}, q, T_heu)
Input:  frequency density (x_i, f_i), 1 ≤ i ≤ m
Output:  a heterogeneous histogram
  j = 0;
  h is empty histogram;
  for (i = 1; j ≤ m)
    j, b = FindLargest(i, q, T_heu);
    append b to h;
    i = j;
  return Compactify(h);
```

Figure 3: Heuristics to construct heterogenous histograms

rameter we still have to fix. We ran experiments varying the maximal degree from 0 to 4 (see technical report). Increasing the degree from 0 to 1 can result in considerable space savings. For example, the space consumption of the histogram for another weather data attribute called *station pressure* drops from 14710 to 5076 bytes, for a maximal q-error of two. Increasing the degree beyond 1 results in at most 10% space savings for our datasets, which we think is not worth while. Thus, we fix the maximal degree to 1 and only give results for this case.

Table 3 contains the results for `HetHeu`. It contains a small table for each dataset considered here. The first column contains the maximal q-error given as a parameter to `HetHeu`. The other columns contain the number of buckets of the histogram constructed, the size of the histograms in bytes, the number of bits needed for a single distinct value (i.e. $\lceil \text{size} * 8/m \rceil$, where m is the number of distinct values), and the CPU time (in seconds) needed to construct the histogram.

We can make the following observations. Slight increases of the maximal q-error typically result in a significant drop of the histogram's size. Sometimes, space consumption increases with higher q-error bounds (see `wtrslp`). This is obviously due to the fact that `HetHeu` is a heuristics. Some datasets (e.g. `ect_usdeur`, `wtrtmp`) seem to be difficult to approximate, since the number of bits they need for each entry of the frequency density is quite high. Others (e.g. `uniprotMW`) are difficult to approximate due to their sheer number of distinct values (72519 in this case). We also observe that the histogram construction times are quite reasonable. By far the most time is needed by `uniprotMW`, which for a maximal q-error of 2 needs about five minutes to construct the histogram. However, we are not worried about that, as the code has not been tuned for efficiency but flexibility, and we know several points where it can be significantly improved.

Taking a look at Table 2, we see the following. Heterogeneous histograms always outperform homogeneous histograms. Depending on the dataset, the differences can be very large (e.g. for `citeseer`, `UniprotMW`) or quite small (e.g. `wtrslp`, `wtrtmp`).

Dataset	HetOpt Size/bytes	HetHeu Size/bytes	HetOpt CPU/h
citeseer	831	1125	88
ecb_usdeur	6587	7639	126
uniprotAA	806	952	39
wtrslp	2908	4039	34
wtrtmp	3536	5210	65

Table 4: Comparison of `HetOpt` and `HetHeu`

Bucket Type	%
`T-?`	44.8
`-Q?`	8.2
`TQ?`	1.0
bucklet-based	5.8
width-based	15.3
q-compression	23.8

Table 5: Bucket type distribution for `HetHeu`

To see how close the heterogenous histograms' sizes constructed by `HetHeu` are to the minimal possible sizes, take a look at Table 4. The first column denotes the dataset. The second and third column show the sizes of the histograms constructed by `HetOpt` and `HetHeu`. We see that the optimal heterogenous histograms are at most about a third smaller than those constructed by `HetHeu`. Thus, the heuristics does not exploit the full potential of heterogeneous histograms. The fourth column contains the runtime of `HetOpt` in hours. Clearly, the runtimes are much too high and render `HetOpt` useless for any practical purpose.

We present some statistics over the bucket types occurring in histograms constructed by `HetHeu` with a maximal q-error of 2. Table 5 gives for each bucket type the percentage of its occurrence.

Note that we cannot draw any conclusion about the relevance of a bucket type from these numbers. A small percentage of occurrence does not mean the bucket type is less useful than others, since the sizes of the buckets of different types may differ vastly. About 31% of all buckets keep the boundary frequency, where not significance exists with respect to the bucket type.

Let us now step back and compare the results for heterogenous histograms with those of commercial systems. Remember that commercial systems virtually have no upper bound on the error they produce. Factors of 100 and more seem to be quite common. In contrast, we can limit the maximal error to an arbitrary number but may pay for it in terms of memory consumption. Commercial systems are clearly willing to spend 3200 bytes. Some allow the user to specify arbitrary upper bounds for memory consumption. If we are willing to spend 10KB of memory, the histograms for five of our six datasets have a maximal q-error of 1.7 and one 5.0. The cost of 10KB of main memory is about 0.0002 cent. Thus, for a hundred relations with a hundred attributes each, some being simple, some tractable or challenging, we need about a dollar of main memory to store histograms which guarantee an upper q-error of two. We believe that this dollar is well invested.

6. CONCLUSION

We have seen that commercial systems create huge errors when estimating the cardinalities of the different query types EMQ, DCT, and RGE. Thus, to show a possibility to limit the q-error to 2 or even below for all query types simultaneously is a major achievement of this paper. This advancement over the state of the art was only possible by giving up the restriction that in every bucket the number of distinct values it contains and their cumulated frequency is stored. In fact, only heterogeneous histograms are able to guarantee low error bounds while at the same time keeping space consumption at a reasonable level.

Future work has to be done to further improve the achievements. As we have seen, the q-optimal heterogeneous histograms sometimes occupy 30% less storage than those constructed by the heuristics. Besides, inventing more bucket types may help to further reduce memory consumption. A larger and more interesting challenge will be to construct multi-dimensional histograms, which can be built simultaneously over two or more attributes. This is an important issue, since correlations between attributes still pose a major problem in the context of cardinality estimation.

Acknowledgment. We thank Uwe Steinel for the administration of the three commercial database systems and Sebastian Sindermann for carrying out the experiments with them. We also thank Simone Seeger for her help preparing the manuscript.

APPENDIX

A. ALGORITHM FOR BEST APPROXIMATION UNDER L_Q

We review a basic algorithm to solve the best approximation problem under l_q. One such algorithm is described in [11]. The one we present here is faster and easier to understand. The presented algorithm is applicable to best approximation problems under l_q by any polynomial. But, for simplicity, we only review the algorithm for best linear approximations under l_q.

Before we start describing the algorithm, we need some preliminaries. A linear polynomial is uniquely determined by a (solvable) system of two equations with two variables, say α and β. We add a third variable λ, which denotes the q-error under which a given set of three points can be best approximated under l_q. As shown in [11], this approximation always exists and is unique.

In order to build a system with three equations and three variables, we make use of a theorem from [11] saying that the errors alternate in sign. That is, for any three points (x_i, y_i) $(1 \leq i \leq 3)$ such that $x_i < x_{i+1}$ $(1 \leq i < 3)$ and the best approximation $\hat{f}$ under l_q of these three points, we either have

$$y_1 \leq \hat{f}(x_1) \wedge y_2 \geq \hat{f}(x_2) \wedge y_3 \leq \hat{f}(x_3)$$

or

$$y_1 \geq \hat{f}(x_1) \wedge y_2 \leq \hat{f}(x_2) \wedge y_3 \geq \hat{f}(x_3).$$

Thus, as we can have $\lambda \geq 1$ or $\lambda \leq 1$, solving the following system of three equations gives us the coefficients α and β of the best linear approximation as well as λ:

$$\begin{aligned} \frac{1}{\lambda}(\alpha + \beta x_1) &= y_1 \\ \lambda(\alpha + \beta x_2) &= y_2 \\ \frac{1}{\lambda}(\alpha + \beta x_3) &= y_3 \end{aligned}$$

Thus, for any three points, we can solve the problem.

In order to generate a solution for an arbitrary set of points, we need another theorem from [11]. Let $X = \{(x_i, y_i)\}$ be the set of points, we want to approximate. Then, the theorem says that there exists an *extremal subset* containing three points of X. This means that there exist three points, such that their best approximation under l_q is the same as the best approximation of the whole set X under l_q.

Roughly, the algorithm finds these three points as follows. It starts with an arbitrary subset of X that contains three points. Then, it calculates their best approximation under l_q. This approximation is then used, to find the point in X for which the approximation generates the largest q-error. This point is then exchanged with one of those contained in the original set of three points. From here, the algorithm iterates, until the q-error cannot be increased by any exchange. In detail, we have

1. Choose arbitrary i_1, i_2, i_3 with $x_{i_1} < x_{i_2} < x_{i_3}$.
2. Calculate the solution for the system of equations. This gives us an approximation function $\hat{f}(x) = \alpha + \beta x$ and λ.
3. Find an x_k for which the deviation $||\hat{f}(x_k)/y_k||_Q$ is maximized. Call this maximal deviation $\lambda_{\max}$.
4. If $\lambda_{\max} - \lambda > \epsilon$ for some small ϵ
   then apply the exchange rule (see below) using x_k and go to step 2.
   (The ϵ is mainly needed for rounding problems with floating point numbers. If they were non-existant, one could choose $\lambda_{\max} \neq \lambda$ as the criterion.)
5. Return α, β, λ.

What is left to be done, is to specify the exchange rule. For given i_1, i_2, i_3 with $x_{i_1} < x_{i_2} < x_{i_3}$ and derived α, β, λ, we try to find new indices j_1, j_2, j_3 by exchanging one of the i_j with k such that λ will be increased. Assume the deviation of the (current) estimate $\hat{f}$ is maximized at some k. Then, we will exchange one of the i_1, i_2, i_3 by k according to the following exchange rule. Define $\hat{f}_i = \alpha + \beta x_i$. Depending on the position of x_k in the sequence i_1, i_2, i_3 and the sign of the residual, we determine the i_j to be exchanged with k:

- $x_k < x_{i_1}$
  if $(\text{sign}(y_k - \hat{f}_k) == \text{sign}(y_{i_1} - \hat{f}_{i_1}))$
  then $j_1 = k, j_2 = i_2, j_3 = i_3$
  else $j_1 = k, j_2 = i_1, j_3 = i_2$
- $x_{i_1} < x_k < x_{i_2}$
  if $(\text{sign}(y_k - \hat{f}_k) == \text{sign}(y_{i_1} - \hat{f}_{i_1}))$
  then $j_1 = k, j_2 = i_2, j_3 = i_3$
  else $j_1 = i_1, j_2 = k, j_3 = i_2$
- $x_{i_2} < x_k < x_{i_3}$
  if $(\text{sign}(y_k - \hat{f}_k) == \text{sign}(y_{i_2} - \hat{f}_{i_2}))$
  then $j_1 = i_1, j_2 = k, j_3 = i_2$
  else $j_1 = i_1, j_2 = i_2, j_3 = k$

- $x_k > x_{i_3}$
  if $(\text{sign}(y_k - \hat{f}_k) == \text{sign}(y_{i_3} - \hat{f}_{i_3}))$
  then $j_1 = i_1, j_2 = i_2, j_3 = k$
  else $j_1 = i_2, j_2 = i_3, j_3 = k$

B. REFERENCES

[1] M. Charikar, S. Chaudhuri, R. Motwani, and V. Narasayya. Towards estimation error guarantees for distinct values. In *Proc. ACM SIGMOD/SIGACT Conf. on Princ. of Database Syst. (PODS)*, pages 268–279, 2000.

[2] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[3] P. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, pages 541–550, 2001.

[4] L. Haas, M. Carey, M. Livny, and A. Shukla. Seeking the truth about ad hoc join costs. *The VLDB Journal*, 6(3):241–256, 1997.

[5] Y. Ioannidis. Universality of serial histograms. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, pages 256–267, 1993.

[6] Y. Ioannidis and S. Christodoulakis. Optimal histograms for limiting worst-case error propagation in the size of join results. *ACM Trans. on Database Systems*, 18(4):709–748, 1993.

[7] Y. Ioannidis and V. Poosola. Balancing histogram optimality and practicality for query result size estimation. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 233–244, 1995.

[8] Y. Ioannidis and V. Poosola. Histogram-based solutions to diverse database estimation problems. *IEEE Data Engineering Bulletin*, 18(3):10–18, Sept 1995.

[9] R. Kooi. *The Optimization of Queries in Relational Databases*. PhD thesis, Case Western Reserve University, 1980.

[10] G. Moerkotte. Profiles for single attributes. Technical Report TR-2010-003, Department of Mathematics and Computer Science, University of Mannheim, Mannheim, Germany, 2010.

[11] G. Moerkotte, T. Neumann, and G. Steidl. Preventing bad plans by bounding the impact of cardinality estimation errors. *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, 2(1):982–993, 2009.

[12] B. Piatetsky-Shapiro and C. Connell. Accurate estimation of the number of tuples satisfying a condition. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 256–276, 1984.

[13] V. Poosola, Y. Ioannidis, P. Haas, and E. Shekita. Improved histograms for selectivity estimates of range predicates. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 294–305, 1996.