Sample-based Distinct Cardinality Estimation for Multiple Attributes in Multi-Dataset Queries

Mehnaz Tabassum Mahin
University of California, Riverside
mehnaztabassum.mahin@email.ucr.edu

Michael J. Carey
University of California, Irvine
mjcarey@ics.uci.edu

Vassilis J. Tsotras
University of California, Riverside
tsotras@cs.ucr.edu

ABSTRACT

Estimating the number of distinct values in an attribute or a set of attributes is one of the classical and open problems of cost-based query optimizers (CBOs). Such estimations can be very difficult to make in the presence of query selection predicates without examining the complete dataset. It becomes even harder for a multi-dataset (i.e., join) query with selection predicates.

Recent advances in CBOs have introduced sample-based approaches, which maintain stored samples on the underlying datasets to improve the accuracy of cardinality and selectivity estimation during query compilation. Leveraging these stored samples, this paper addresses the important yet challenging problem of estimating the number of distinct values in an attribute or a set of attributes in a multi-dataset query. We refer to our proposed sample-based approach as the *MAMD* (Multi-Attribute, Multi-Dataset) approach. The MAMD approach works for join queries with or without selection predicates and is also effective for estimating the number of distinct values in single-dataset queries. We present an experimental evaluation of the proposed MAMD approach with synthetic and real-world datasets, namely the TPC-H and the IMDB benchmark datasets. We demonstrate how it can estimate the number of distinct values with moderately low relative errors and with low storage overhead and execution time. We also investigate how the MAMD approach performs when we scale up the size of the database.

PVLDB Reference Format:
Mehnaz Tabassum Mahin, Michael J. Carey, and Vassilis J. Tsotras. Sample-based Distinct Cardinality Estimation for Multiple Attributes in Multi-Dataset Queries. PVLDB, 19(6): 1115 - 1127, 2026.
doi:10.14778/3797919.3797922

1 INTRODUCTION

Query optimizers are an important component of database systems. A cost-based optimizer (CBO) is a part of most current database management systems [1–3]. Unlike rule-based optimizers, CBO uses metadata and data statistics, such as data distribution, dataset sizes, index information, etc., to evaluate multiple query execution plans and determine the plan with the lowest estimated cost. In doing so, CBO considers different join orders and chooses the optimal join order based on its cost estimation.

Proceedings of the VLDB Endowment, Vol. 19, No. 6 ISSN 2150-8097.
doi:10.14778/3797919.3797922

CBO relies on efficient cardinality estimation, traditionally using histogram-based statistics. While such statistics offer reasonable accuracy for range and point queries over numeric attributes, they fall short when dealing with composite keys, non-numeric attributes, and queries involving complex predicates such as LIKE, spatial operations, or user-defined functions (UDFs). To address these challenges, modern data-intensive systems [1, 28] have recently adopted *sample-based approaches* in their cost-based optimizers (CBOs), particularly for selectivity and cardinality estimation. These CBOs maintain *stored samples* on the underlying datasets and query them to derive the estimates at compilation time.

To leverage the more informative *stored samples* maintained by CBO, in this paper, we focus on one of the classical problems in cardinality estimation: *distinct cardinality estimation*, i.e., estimating the number of *distinct values* in an attribute or a set of attributes. This is important for determining the expected result size of a GROUP BY or a DISTINCT operator. Distinct cardinality estimation has also been used for fast approximate query answering [15] and online aggregation [21].

Traditionally, cardinality estimation deals with estimating the ***number of tuples*** that satisfy selection predicates over a single dataset, or join predicates among multiple datasets. In contrast, estimating the ***number of distinct values*** in a set of attributes is inherently difficult without exploring the full dataset. It can be even more difficult in the presence of selection predicates. For example, in the following query, the number of distinct values in the combination of `(T.a1, T.a2)` depends on how many tuples satisfy the selection predicate on attribute `T.a4`:

```
SELECT T.a1, T.a2, avg(T.a3)
FROM T WHERE T.a4 = c
GROUP BY T.a1, T.a2;
```

A good distinct cardinality estimation for such a query would enable CBO to choose a cost-effective algorithm between hash-based and sort-based options to implement the GROUP BY operator.

Estimating the number of distinct values for a *single attribute* in a dataset has been well-addressed in the database literature. These distinct estimators are either adopted from the statistics literature to database settings [23, 24, 33] or proposed directly for database frameworks [7, 17]. Several research studies [14, 42] have focused on estimating the number of distinct values for a *set of attributes* in a dataset. Among existing works, ***sample-based*** approaches are widely adopted since they can handle arbitrary selection predicates.

To the best of our knowledge, there is no previous work on estimating the number of distinct values in *multi-dataset queries* (i.e., join queries). This problem is challenging since examining the full result of a join query is not feasible in practice at optimization time. Even generating a random sample of the results of a single join, known as "join sampling", is inherently difficult, since the sampling

operator cannot be pushed through a join [8]. Prior related research demonstrates the challenges of "join sampling" for joins over multiple datasets, even with a fixed join order in a join tree [10, 43]. Though "join sampling" could capture the attribute-level correlations for a specific *multi-dataset* query with a given join order and attribute combinations, generating and analyzing such samples at query optimization time is not feasible for CBO, which must handle a wide range of queries. Moreover, CBO faces further challenges: (i) it needs to consider join sampling for all possible join orders examined, and (ii) selection predicates in such join queries make distinct cardinality estimation even more difficult.

In this paper, we focus on the distinct cardinality estimation problem and propose a sample-based approach for estimating the number of distinct values in *a set* (combination) *of attributes* in *multi-dataset queries*. We also address the estimation problem even in the presence of *selection predicates* ranging across multiple attributes. The proposed approach examines *stored samples* of each of the query's individual datasets, i.e., it differs from "join sampling". Since join-level samples depend on query-specific join graphs and predicates, CBO cannot materialize join samples during optimization and our approach leverages *stored samples* (pre-collected by the system before optimization). As a result, our approach requires low storage overhead and scales with increasingly large datasets.

One important application of distinct cardinality estimation is to make a better choice between hash-based and sort-based GROUP BY algorithms, which helps avoid slow query plans. Both algorithms are relatively tolerant of moderate errors in distinct cardinality estimation, as CBO needs to decide whether the estimated distinct cardinality is small enough (e.g., can be fit in memory) to consider the hash-based (or large enough to favor the sort-based) algorithm. For example, let the actual number of distinct values in `T.a` be 60k, the available memory for grouping be 180 MB, and the hash table entry size be 2000 bytes. If the estimated number of distinct values is 65k, the estimated hash table size is 130 MB, so CBO chooses the hash-based algorithm. On the other hand, if the estimate is 100k, the estimated hash table size becomes 200 MB, exceeding the memory budget, and CBO switches to the sort-based algorithm. Since sorting 100k rows is significantly slower than hashing 60-65k groups, this choice affects the query plan's performance. Furthermore, it has been shown [7] that no cardinality estimator can guarantee small error (or, high accuracy) across all attribute-value distributions unless it examines a large fraction of the datasets. Therefore, in this paper, we focus on reducing the relative error in the estimated number of distinct values in a set of attributes in join queries instead of aiming to deliver accuracy guarantees.

To evaluate the effectiveness of our proposed sample-based approach, we use Apache AsterixDB as our implementation platform, which already supports basic sample-based cardinality estimation. It maintains a fixed-size random sample for each dataset and examines these samples during query compilation. We leverage this existing infrastructure and integrate our solution naturally into the query optimization workflow of AsterixDB.

The remainder of the paper is organized as follows: Section 2 discusses related work on cardinality estimation and how our approach differs from previous research. We outline some preliminaries in Section 3 and provide an overview of the proposed sample-based approach in Section 4. Section 5 describes how the stored samples are used to estimate the distinct cardinality in *a set of attributes* in *multi-dataset queries*. Experimental results and their analysis are presented in Section 6. Finally, Section 7 concludes the paper and discusses some directions for future research.

2 RELATED WORK

Cardinality estimation is a fundamental part of query optimization in modern database systems. Estimating the selectivity of predicates has been extensively studied in the literature [16, 18, 25, 28]. A thorough survey on cardinality estimation methods and their application in query optimization appears in [9].

Traditionally, database systems collect and store metadata and the underlying data statistics. Several approaches (e.g., sampling-based, learning-based, synopsis-based, etc.) have been proposed to examine each of a query's datasets along with the stored statistics and derive cardinality estimates. While sampling-based approaches are widely adopted in existing research [7, 17, 18, 38] for estimation problems, modern systems [1, 4, 28] increasingly focus on *stored samples*, as querying stored samples can handle more complex predicates (e.g., LIKE predicates) than range predicates. Additionally, because the stored samples are fixed in size, acquired in advance, and accessed at compile time, they can handle increasingly large datasets. Though learning-based approaches [12, 20, 26, 29–31, 36, 40, 41] have recently been considered for capturing the data distribution, a recent study [39] discussed how learning-based methods could be affected by fast data updates and changes in correlation, skewness, or domain size. Additionally, they can suffer from high training and inference costs, especially in Big Data analytics systems. In practice, learning-based methods require access to user data and historical query workloads for training, which commercial DBMSs cannot collect or store due to privacy and operational constraints. Synopsis-based research [9] has mainly focused on histogram-based [35, 42] and sketch-based [13, 14, 22] approaches. For arbitrary combinations of attributes, these approaches require maintaining multi-dimensional histograms and multiple sketches. However, it is not feasible to maintain a potentially exponential number of histograms and sketches without prior knowledge of combinations of attributes; at the same time, it is also expensive to maintain such synopses in the event of updates and deletions. Since sampling-based approaches can handle arbitrary attributes and predicates without expensive full scans and update operations, they remain the popular choice among researchers for estimating the number of distinct values in an attribute or a set of attributes.

With respect to distinct cardinality estimation, previous research has addressed the problem within a single dataset. Estimating the number of distinct values for a *single attribute in a single dataset* is well-studied in the statistics literature (a detailed survey appears in [6]) and in the database literature [17, 19, 23, 24, 33]. Several sampling techniques to estimate the distinct cardinality in a dataset are proposed in [4, 32]. Haas *et al.* [17] presented an extensive study on existing distinct estimators and proposed several sampling-based estimators for an attribute in a dataset, one of which is a hybrid estimator. The hybrid estimator uses a smoothed jackknife estimator for uniform to moderately skewed data and the Shlosser estimator [38] to characterize the degree of data skew. Charikar *et*

al. [7] established a powerful negative result revealing the inherent difficulty of distinct estimators and subsequently, proposed a sampling-based heuristic estimator for a single attribute that adapts to the input data distribution obtained through random samples. Though their proposed heuristic estimator performs better with low-skew data, it cannot give strong guarantees on distribution-independent worst-case error.

There is also research [14, 42] on estimating the number of distinct values in *a set* (combination) *of attributes in a single dataset*. Yu *et al.* [42] proposed a distinct cardinality estimator for a set of attributes that employs existing knowledge about the underlying data maintained in the database catalog. First, they discussed a scenario where exact frequency information is available in the catalog and can be used as probabilities to estimate the number of distinct value combinations. However, exact frequency information for all possible attributes cannot be maintained in practical cases. Then, they devised histogram-based estimators to provide approximate estimates where partial information is available. In recent work, Freitag and Neumann [14] proposed an estimation framework that combines sketches and samples to estimate the number of arbitrary attribute combinations in a dataset with high accuracy. It maintains sketched information over individual attributes and uses random sampling to capture the correlation bias between attributes.

We are aware of no existing work that addresses the important and challenging problem of estimating the number of distinct values in *a set of attributes* in *multi-dataset queries* (i.e., join queries). In modern large-scale database systems, collecting histograms or sketches for all possible attributes in all datasets distributed over multiple partitions and consistently maintaining them in the system catalog is not feasible. Also, capturing the correlation among attributes across multiple datasets is nearly impossible if the attributes are not primary or foreign keys. This leads us to propose a sample-based distinct cardinality estimation approach, namely the *MAMD* (Multi-Attribute, Multi-Dataset) approach, that maintains small independent fractions of all datasets, i.e., *sample datasets*, and utilizes the actual frequency information in the sample datasets to estimate distinct cardinalities during query optimization.

3 PRELIMINARIES

In this section, we outline some preliminaries used in the proposed sample-based distinct cardinality estimation approach.

3.1 Sampling Method and Cardinality

To generate a *sample dataset*, we consider the widely used *simple random sampling without replacement* approach [27]. Here, each tuple from the full dataset is selected with *equal probability* and each successive tuple is chosen uniformly at random from the remaining unselected tuples, ensuring that all possible samples of the specified size are equally likely.

Selection Cardinality Estimation. Sample-based selection cardinality estimation approximates the number of tuples satisfying selection predicates by examining stored sample datasets. In systems like Apache AsterixDB, users can run ANALYZE statements to refresh the sample, during which the base dataset is analyzed, and randomly selected tuples are stored as its sample dataset. At query compilation time, the cost-based optimizer (CBO) splits a multi-dataset query into single-dataset queries, with associated selection predicates, to run against the samples and estimates the selectivities. This allows CBO to handle a wide range of predicates, from simple constants to more complex ones, such as LIKE predicates, functions, or nested fields. Figure 1 illustrates how AsterixDB's CBO will query the sample datasets *cs* and *os* of the `Customer` and `Orders` datasets from TPC-H, respectively, during compilation.

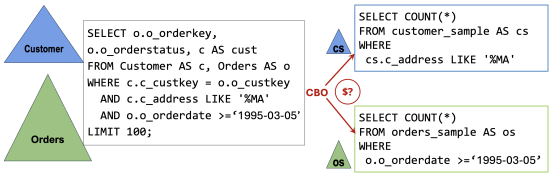


Figure 1: Cost-based optimization using samples.

Join Cardinality Estimation. We follow the usual textbook formula [37] to estimate the join cardinality. Let L and R be the two datasets participating in a join with the join predicate $L.a = R.b$, where $|L|$ and $|R|$ are the cardinalities of the respective datasets. If there are no selection predicates, and $|L.a|$ and $|R.b|$ are the number of distinct values in the corresponding attributes, then the estimated result size of the join, $|L \bowtie R| = \frac{|L||R|}{\max(|L.a|,|R.b|)}$ and the join selectivity, $sel(|L \bowtie R|) = \frac{1}{\max(|L.a|,|R.b|)}$.

Let p be the selection predicates on the join, and $|L|_p^{est}$ and $|R|_p^{est}$ be the corresponding selected dataset cardinality estimated from examining the sample datasets. Now, considering the selection predicates p, the estimated cardinality of the join $N_{\bowtie}^{est}$ becomes,

$$N_{\bowtie}^{est} = sel(|L \bowtie R|) \times |L|_p^{est} \times |R|_p^{est}.$$

Can we estimate the distinct cardinality easily? Consider estimating the number of distinct values in an attribute $T.a$ of a dataset T. Let $|T|$ be the dataset size, $|T_S|$ be the size of the sample dataset, and d be the number of distinct $T.a$ values in the sample dataset T_S. As a first attempt, one may be tempted to estimate the number of distinct $T.a$ values in T using a scale-up factor: $\frac{d}{|T_S|} \times |T|$. This scale-up factor will work if $T.a$ is an attribute like `name` or `email_address` and approximately scales with the primary key of T. However, if $T.a$ is an attribute like `gender`, `rank`, or `zip_code` with a precise (limited) number of distinct values in T, the scale-up factor can cause the estimated value to be either too high or too low.

This example shows the need to consider an estimator that can observe the *stored sample* (i.e., the sampled tuples) and using the distinct value data distribution in the sample dataset, estimate the distinct cardinality in the full dataset.

3.2 Method-of-Moments Estimator

A simple estimator, adopted from the statistical literature [5] and known as the *Method-of-Moments* (MM) estimator [17], can address the above issue. It takes as inputs a sample size n and the number of distinct values d in an attribute observed from the sample, and computes the estimated number of distinct values D in the original dataset. In particular, D is the solution of the following equation:

$$d = D(1 - e^{-n/D}) \quad (1)$$

Equation (1) can be solved numerically using the Newton-Raphson method, and the final value of D is returned as the number of distinct values in the attribute.

This estimator assumes that the sample dataset is taken from an infinite population and the frequencies of the distinct values are all equal, which is equivalent to the *sampling* ***with*** *replacement* approach. This differs from our *sampling* ***without*** *replacement* approach. Nevertheless, equation (1) depends only on the sample size and the number of distinct values in the sample, allowing us to consider it as a base estimator and adapt it to our setting.

Note that the MM estimator cares only about the distinct values observed in a sample taken from a set of tuples (independent of whether these values come from a single attribute or a combination of attributes). As a result, we can adapt this estimator for *a set of attributes* in a *single dataset*, since the combined value from a set of attributes can be collectively viewed as a single value. Consider a dataset T, its sample dataset T_S, and a set of attributes $T.\mathbf{a} = (a_1, a_2, \ldots, a_i)$. Note that a given query can have selection predicates on *any* attributes in T. Given a query, if p denotes all the query's selection predicates on T, let n_p be the number of tuples T_{S_p} from the sample dataset T_S that satisfy the selection predicates p (i.e., $n_p = |T_{S_p}|$). Similarly, let d_p be the number of distinct values in the $T.\mathbf{a}$ set of attributes from T_{S_p}, and let D^{est} denote the estimated number of distinct value combinations of $T.\mathbf{a}$ (satisfying p) within T. Then we can rewrite the above equation as follows:

$$d_p = D^{est}(1 - e^{-n_p/D^{est}}) \tag{2}$$

To solve for D^{est} in equation (2) numerically using the Newton-Raphson method, we start with d_p as the initial guess of D^{est} (since d_p is the minimum number of distinct values), calculate the next guess of D^{est} using equation (2) and its derivative, and continue approximating the improved guess of D^{est} iteratively until it converges to a sufficiently precise value. Clearly, we can also use equation (2) for estimating the number of distinct values in *a single attribute* in a single-dataset query with selection predicates.

4 SAMPLE-BASED APPROACH

We first present an overview of the proposed approach and then introduce compile-time sampling queries to examine the sample of each dataset in a given query in Section 4.2.

4.1 Approach Overview

This section summarizes the proposed sample-based distinct cardinality estimation approach for estimating the number of distinct values in *an attribute* or *a set of attributes*. We consider two different kinds of queries:

- **Single-dataset query:** Given a dataset T, estimate D^{est}, i.e., the number of distinct value combinations in a set of attributes $T.\mathbf{a} = (a_1, a_2, \ldots, a_i)$.
- **Multi-dataset query:** Given m datasets $T1, T2, \ldots, Tm$, estimate D^{est}, i.e., the number of distinct value combinations in a set of attributes across multiple joined datasets. Let $T1$ and $T2$ be two datasets in a simple multi-dataset query, then $(T1.\mathbf{a}, T2.\mathbf{b}) = (a_1, \ldots, a_i, b_1, \ldots, b_j)$ represents the set of attributes, where $a_i \in T1$ and $b_j \in T2$ for $i, j = 1, 2, \ldots$.

Note that both *single-dataset* and *multi-dataset* (i.e., join) queries can have selection predicates, and each join query will also have join predicates. For simplicity, we only consider equi-join queries.

In the proposed sample-based approach, we collect and store *sample datasets* for *all* datasets before executing any queries, irrespective of query predicates and distinct attribute combinations. This step is (periodically) performed by the system users (e.g., via an ANALYZE statement run by a DBA).

Given a *single-dataset* query, we examine its *sample dataset* by executing *sampling queries* during **compilation time** (see Section 4.2), collect n_p and d_p values, feed them to equation (2), and estimate the number of distinct values D^{est} in a set of attributes.

Given a *multi-dataset* query, we first parse the given query into multiple single-dataset queries that include the corresponding selection predicates and GROUP BY clauses, and then generate sampling queries for each single-dataset query by replacing the dataset name with the corresponding sample dataset name. After executing all sampling queries, we have the corresponding n_p and d_p values for each of the query's datasets. Finally, we estimate D^{est} in a set of attributes in the given multi-dataset query based on the proposed *MAMD* approach (see Section 5).

4.2 Compile-Time Sampling Queries

Given a query, we execute compile-time selection queries to collect the n_p and d_p values from each sample dataset. We refer to each such compile-time selection query as *a sampling query*. Each sampling query will include the corresponding selection predicates. Consider the following query example on dataset T:

```
SELECT T.a1, T.a2, avg(T.a3)
FROM T WHERE T.a4 = c
GROUP BY T.a1, T.a2;
```

For this query, we will execute the following sampling query on the corresponding sample dataset T_S.

```
SELECT TS.a1, TS.a2, count(*)
FROM TS WHERE TS.a4 = c
GROUP BY TS.a1, TS.a2;
```

Here, the sampling query includes the GROUP BY clause of the given query and returns the cardinality of each distinct value combination in `(TS.a1, TS.a2)` from T_S that satisfies the selection predicate `TS.a4 = c`. Assume that the sampling query returns $(v1, 4)$, $(v2, 1)$, $(v3, 4)$, and $(v4, 3)$, where, $v1, v2, v3, v4$ are the distinct value combinations in `(TS.a1, TS.a2)`, with their respective cardinalities. Since each of $v1$ and $v3$ appears four times in T_S, we can transform $(v1, 4)$ and $(v3, 4)$ to a count-occurrence pair $(2, 4)$. Similarly, we get $(1, 1)$ from $(v2, 1)$ and $(1, 3)$ from $(v4, 3)$. Thus, we transform the cardinalities into a vector of count-occurrence pairs: $[(2, 4), (1, 1), (1, 3)]$.

Let $O_{T_S} = [(\delta_{t_1}, o_{t_1}), \ldots, (\delta_{t_k}, o_{t_k})]$ be the *count-occurrence vector* of k pairs collected from T_S, where each k^{th} pair (δ_{t_k}, o_{t_k}) represents the fact that each of the δ_{t_k} distinct value combinations appears o_{t_k} times in the sample dataset T_S. Given O_{T_S}, we can derive $d_p = \sum_{\forall k} \delta_{t_k}$ and $n_p = \sum_{\forall k}(\delta_{t_k} \times o_{t_k})$. We will employ the *count-occurrence vector* O_{T_S} in the proposed MAMD approach (see Sections 5.3 and 5.4).

When considering a *multi-dataset* query, there is a special case in which there are no attributes from one of the datasets in the

GROUP BY clause of a multi-dataset query. Consider the following multi-dataset query:

```
SELECT T.a1, T.a2, avg(T.a3)
FROM T JOIN R ON T.a5 = R.b1
WHERE T.a4 = c AND R.b2 = e
GROUP BY T.a1, T.a2;
```

For this query, the sampling query on the dataset T remains the same as discussed before. However, since there are no attributes from the dataset R in the GROUP BY clause, we execute a modified sampling query on the corresponding sample dataset R_S as follows (i.e., without the GROUP BY clause):

```
SELECT count(*) FROM R_S
WHERE R_S.b2 = e;
```

This sampling query returns n_p, the number of the tuples in the sample dataset of R that satisfy the predicate (R_S.b2 = e), which is eventually used during the join cardinality estimation for the given multi-dataset query, as discussed in Section 3.1.

5 DISTINCT CARDINALITY ESTIMATION

We now present the details for estimating the number of distinct values in *a set of attributes* in a *multi-dataset* (i.e., join) query.

Evaluation Metric: To evaluate the effectiveness of the proposed estimation approach, we will compute the relative error between the estimated distinct cardinality D^{est} and the actual distinct cardinality D^{act}, satisfying the selection predicates p. Considering the full cardinality N^{act} (i.e., *excluding* p), the relative error can be expected as:

$$RE(\%) = \frac{|D^{act} - D^{est}|}{N^{act}} \times 100$$

where, N^{act} is the actual full join cardinality in a *multi-dataset* query or the actual dataset size in a *single-dataset* query. However, note that N^{act} can be very large, depending on the number of datasets and the cardinality of each dataset participating in a multi-dataset query. It can be even larger and unpredictable if both attributes in a given join predicate are foreign keys (especially, for large-scale real-world datasets, which may not be uniformly distributed).

For example, let us consider the largest datasets, `cast_info` and `movie_info`, from the real-world Internet Movie Database (IMDB) [28], with 36 million and 15 million tuples, respectively. If these datasets participate in a join on attribute `movie_id` (which is a **foreign key** in both datasets), the actual join cardinality becomes 460 million, leading $RE(\%)$ to be very low (approximately zero), even with a high absolute error $|D^{act} - D^{est}|$. As a result, the metric $RE(\%)$ can mislead the estimation performance in such scenarios.

This issue leads us to consider a more natural choice to measure the relative error by considering the number of tuples that satisfy the selection predicates p of the given query. Thus, in this paper, we define the relative error as:

$$RE_p(\%) = \frac{|D^{act} - D^{est}|}{N_p^{act}} \times 100 \quad (3)$$

where, N_p^{act} is the actual join cardinality in a *multi-dataset* query or the actual dataset size in a *single-dataset* query, satisfying selection predicates. Using $RE_p(\%)$ can provide a crucial measure to understand the ***overall performance*** of the *sample-based* distinct cardinality estimator with queries with various selection predicates.

5.1 A Naive Approach to Estimation

We first present a naive approach for estimating the number of distinct values in a set of attributes in *multi-dataset* (i.e., join) queries.

For simplicity, consider a 2-dataset join query. Let L and R be left and right datasets participating in the 2-dataset join and $(L.\mathbf{a}, R.\mathbf{b}) = (a_1, \ldots, a_i, b_1, \ldots, b_j)$ be a set of attributes in the GROUP BY clause of the given query, where $a_i \in L$ and $b_j \in R$ for $i, j = 1, 2, \ldots$. We aim to estimate D^{est}, the number of distinct $(L.\mathbf{a}, R.\mathbf{b})$ values in $\sigma_p(L \bowtie R)$, where p represents the selection predicates.

As described in Section 4, compile-time sampling queries with respective selection predicates for L and R can be generated and executed to get the corresponding n_p and d_p values from both sample datasets. Then, the number of distinct values in $L.\mathbf{a}$ and $R.\mathbf{b}$ can be estimated separately using equation (2).

Let D_L^{est} and D_R^{est} be the number of distinct values in $L.\mathbf{a}$ and $R.\mathbf{b}$, respectively. Let $N_{\bowtie}^{est}$ be the estimated join cardinality of $\sigma_p(L \bowtie R)$, satisfying the selection predicates.

With the calculated D_L^{est} and D_R^{est}, a heuristic can be applied to propagate the distinct cardinality of the corresponding datasets in a 2-dataset join. The heuristic scales up the maximum of D_L^{est} and D_R^{est} by the ratio of the estimated join cardinality $N_{\bowtie}^{est}$ to the cardinality of the respective dataset, either $|L|$ or $|R|$. Equation (4) depicts the resulting computation for the number of distinct values D^{est} in $(L.\mathbf{a}, R.\mathbf{b})$:

$$D^{est} = (D_L^{est} > D_R^{est}) \; ? \; \left(D_L^{est} \times \frac{N_{\bowtie}^{est}}{|L|}\right) : \left(D_R^{est} \times \frac{N_{\bowtie}^{est}}{|R|}\right) \quad (4)$$

In the above equation, if D^{est} becomes larger than $N_{\bowtie}^{est}$, it is replaced with the estimated join cardinality $N_{\bowtie}^{est}$.

Furthermore, there can be a special case where the GROUP BY clause of the given query contains only $L.\mathbf{a}$ (or, $R.\mathbf{b}$) instead of $(L.\mathbf{a}, R.\mathbf{b})$. In this case, all distinct values of $L.\mathbf{a}$ (or, $R.\mathbf{b}$) are likely to be present after the join operation. In such cases, we use $D^{est} = D_L^{est}$ (or, $D^{est} = D_R^{est}$) without applying equation (4).

In the general case, consider a multi-dataset query consisting of m datasets. Let Q be a query tree with m-datasets $\{T1, T2, \ldots, Tm\}$. Assume that the nodes in Q are either join or leaf nodes (i.e., datasets). We first compute D_{Ti}^{est} for each Ti dataset, where $1 \leq i \leq m$, using equation (2). If a join node has two leaf nodes as inputs, D^{est} for the join node is computed using equation (4) and is propagated to the next higher-level join node. If a join node has a non-leaf child node, a similar computation as equation (4) is followed, except that $|L|$ or $|R|$ is replaced with the estimated join cardinality of the corresponding non-leaf join node. Finally, D^{est} of the highest join node is returned as the output.

A Running Example: Consider the join query shown below, which uses two TPC-H benchmark datasets, namely `LineItem` and `Orders`. Assume the scale factor of TPC-H to be 1.0. For this scale factor, the cardinality of `LineItem` and `Orders` are 6001215 and 1500000, respectively.

```
SELECT o.o_orderpriority, l.l_partkey,
       COUNT(*) as order_count
FROM Orders o, LineItem l
WHERE l.l_orderkey = o.o_orderkey
  AND l.l_commitdate < l.l_receiptdate
  AND o.o_orderdate >= '1993-07-01'
```

```
    AND o.o_orderdate < '1993-10-01'
  GROUP BY o.o_orderpriority, l.l_partkey;
```

Table 1 shows returned values after executing sampling queries and corresponding estimated distinct values obtained from equation (2).

Table 1: Obtained values for the running example

Dataset	n_p	d_p	Est. distinct cardinality	Dataset cardinality	Est. join cardinality
`LineItem`	691	690	238510	6001215	139455
`Orders`	38	5	5	1500000	

Here, `LineItem` is the left dataset (i.e., $|L| = 6001215$), the number of tuples from the sample dataset of `LineItem` satisfying the selection predicates is $n_p = 691$, the number of distinct values in the attribute `l.l_partkey` from the sample dataset is $d_p = 690$, and the estimated number of distinct values in `l.l_partkey` is $D_L^{est} = 238510$. Using the naive approach, the estimated number of distinct values from equation (4) becomes: $D^{est} = 238510 \times \frac{139455}{6001215} = 5542$, whereas the actual distinct cardinality is $D^{act} = 134942$. As a result, the corresponding relative error $RE_p(\%)$ is:

$$RE_p(\%) = \frac{|134942 - 5542|}{144869} \times 100 = 89.32\%$$

This high relative error of 89.32% reflects a substantial deviation from the actual cardinality, highlighting the need for improvement in the estimation process. This motivates the proposed approach, as introduced in Section 5.2.

5.2 The MAMD Approach

As the example above shows, the heuristic technique of the naive approach can result in a relatively high error even for a 2-dataset join query. In this section, we propose a sample-based approach based on probability theory to estimate the number of distinct values D^{est} in *a set of attributes* in *multi-dataset* queries. We refer to this sample-based approach as the *MAMD* (Multi-Attribute, Multi-Dataset) approach.

The naive approach employs the sample dataset size n_p and the number of distinct values d_p in the sample to estimate D^{est}. However, the sampling query with the corresponding GROUP BY clause (taken from the given query; see Section 4.2) returns an extra piece of information, i.e., the cardinality of each distinct value in the sample dataset. Using this, we can derive the *count-occurrence vector*. MAMD utilizes the count-occurrence vector along with n_p and d_p to estimate the distinct cardinality D^{est}. In particular, to compute D^{est}, MAMD uses a distinct cardinality estimator equation proposed in [42] (namely, Theorem 3) for *a set of attributes* in *a single dataset*. Below, we describe that equation, and in Section 5.3, we show how it can be extended to a multi-dataset scenario.

Let $\mathbf{A} = (A_1, A_2, \ldots, A_i)$ be a set of attributes in a single dataset, D_i be the number of distinct values in each $A_i \in \mathbf{A}$, and N be the dataset size. Also, let $\mathbf{V}$ be the set of all distinct value combinations in $\mathbf{A}$, $f_\mathbf{v}$ be the probability of occurrence of a specific value combination $\mathbf{v} \in \mathbf{V}$ such that $\sum_{\mathbf{v}\in\mathbf{V}} f_\mathbf{v} = 1$, and M be the *maximum* possible number of distinct value combinations, i.e., $M = \min\{\prod_{i=1}^{|\mathbf{A}|} D_i, N\}$. The distinct cardinality D^{est} for the set of attributes $\mathbf{A}$ in a single dataset is given by the following equation (proposed in [42]):

$$D^{est} = M - \sum_{\mathbf{v}\in\mathbf{V}} (1 - f_\mathbf{v})^N \tag{5}$$

Equation (5) is derived based on probability theory under the assumptions that *the data distributions of individual attributes are independent and that the occurrences of individual distinct value combinations are independent.*

Note that, assuming independent data distributions in different attributes, trivial bounds of D^{est} are: $\max\{D_1, D_2, \ldots, D_{|\mathbf{A}|}\}$ (*lower bound*) and $M = \min\{\prod_{i=1}^{|\mathbf{A}|} D_i, N\}$ (*upper bound*). Depending on the dataset size N, the number of distinct values $D_i, \forall A_i \in \mathbf{A}$, and the number of attributes $|\mathbf{A}|$, these bounds can be far apart, whereas D^{act} can be close to either of these bounds or far from them.

To adapt equation (5) to multi-dataset queries, we modify it according to the information collected from samples by executing compile-time sampling queries. We proceed to describe MAMD for 2-dataset joins, followed by multi-dataset joins in Section 5.4.

5.3 MAMD for 2-dataset Joins

Similar to Section 5.1, let L and R be the left and right datasets participating in a 2-dataset join and $(L.\mathbf{a}, R.\mathbf{b}) = (a_1, \ldots, a_i, b_1, \ldots, b_j)$ be a set of attributes in the GROUP BY clause of a given query, where $a_i \in L$ and $b_j \in R$ (for $i, j = 1, 2, \ldots$). Also, let D_L^{est} and D_R^{est} be the number of estimated distinct values in $L.\mathbf{a}$ and $R.\mathbf{b}$, respectively.

By executing compile-time sampling queries (Section 4.2) for both L and R, we calculate D_L^{est} and D_R^{est} using equation (2). Then, the number of *maximum* possible distinct value combinations in $(L.\mathbf{a}, R.\mathbf{b})$ becomes $M = D_L^{est} \times D_R^{est}$. Since the dataset cardinality in equation (5) now comes from the result of a join, we replace N by the estimated join cardinality $N_{\bowtie}^{est}$ (satisfying selection predicates). However, the probability values $f_\mathbf{v}, \forall \mathbf{v} \in \mathbf{V}$ are not readily available from the sample datasets. As a result, we consider a heuristic to compute the *estimated count-occurrence vector* for each dataset (Section 5.3.1) and then derive their *estimated frequency vectors* (Section 5.3.2). Using the estimated frequency vectors, we calculate the probability values for each distinct value combination.

5.3.1 Estimated Count-Occurrence Vector.

Algorithm 1 describes how the *estimated count-occurrence vector* for a dataset is computed from its sample dataset. Given a dataset T and its sample dataset T_S, let D_T^{est} be the estimated distinct values, $|T|_p^{est}$ be the estimated number of tuples in T satisfying the selection predicates, and $O_{T_S} = [(\delta_{t_1}, o_{t_1}), \ldots, (\delta_{t_k}, o_{t_k})]$ be the count-occurrence vector obtained from T_S.

The GetEstimatedVector procedure takes n_p and d_p values along with D_T^{est}, $|T|_p^{est}$, and O_{T_S} as inputs. First, it estimates the number of tuples τ_T in T containing the distinct values that appeared in T_S (line 7). Then, it calls the GetScaledUpVector function (lines 16–25) to scale up o_{t_k} by $\frac{\tau_T}{n_p}$ for the k^{th} count-occurrence pair $(\delta_{t_k}, o_{t_k}) \in O_{T_S}$ and build the *estimated count-occurrence vector* $O_T^{est} = [(\delta_{t_1}, o_{t_1}^{est}), \ldots, (\delta_{t_k}, o_{t_k}^{est})]$. Next, if $D_T^{est} > d_p$ and if there are remaining tuples $\Delta_\tau = (|T|_p^{est} - \sum_{i=1}^{k} (o_{t_i}^{est} \times \delta_{t_i}))$, the procedure calls the GetRestUniformVector function (lines 26–36) to *uniformly* distribute the remaining $(D_T^{est} - d_p)$ distinct values over the

Algorithm 1 Compute an Estimated Count-Occurrence Vector

1: **procedure** GetEstimatedVector($n_p, d_p, D_T^{est}, |T|_p^{est}, O_{T_S}$)
2: n_p : number of tuples in T_S
3: d_p : number of distinct values from T_S
4: D_T^{est} : estimated distinct cardinality
5: $|T|_p^{est}$: estimated number of tuples in T
6: $O_{T_S} = [(\delta_{t_1}, o_{t_1}), \ldots, (\delta_{t_k}, o_{t_k})]$: count-occurrence vector
7: $\tau_T \leftarrow \frac{d_p}{D_T^{est}} \times |T|_p^{est}$
8: $O_T^{est}, \tau' \leftarrow$ GetScaledUpVector(n_p, O_{T_S}, τ_T)
9: $\Delta_\tau \leftarrow |T|_p^{est} - \tau'$ ▷ Δ_τ: number of remaining tuples
10: **if** $(D_T^{est} - d_p) > 0$ **then**
11: $O' \leftarrow$ GetRestUniformVector($(D_T^{est} - d_p), \Delta_\tau$)
12: $O_T^{est} \leftarrow [O_T^{est}, O']$
13: **end if**
14: **return** O_T^{est} ▷ est. count-occurrence vector for T
15: **end procedure**

16: **function** GetScaledUpVector(n_p, O_{T_S}, τ_T)
17: τ_T : est. number of tuples with sampled distinct values
18: $\tau' \leftarrow 0, O_T^{est} \leftarrow []$
19: **for each** $(\delta_{t_i}, o_{t_i}) \in O_{T_S}$ **do** ▷ iterate i from 1 to k
20: $o_{t_i}^{est} \leftarrow \left\lfloor o_{t_i} \times \frac{\tau_T}{n_p} \right\rceil$ ▷ rounded to the nearest integer
21: $\tau' \leftarrow \tau' + o_{t_i}^{est} \times \delta_{t_i}$
22: $O_T^{est} \leftarrow [O_T^{est}, (\delta_{t_i}, o_{t_i}^{est})]$
23: **end for**
24: **return** O_T^{est}, τ'
25: **end function**
26: **function** GetRestUniformVector($\Delta_{D^{est}}, \Delta_\tau$)
27: $\Delta_{D^{est}}$: number of remaining distinct values
28: Δ_τ : number of remaining tuples
29: $round \leftarrow \left\lfloor \frac{\Delta_\tau}{\Delta_{D^{est}}} \right\rceil, c_1 \leftarrow \Delta_\tau - round \times \Delta_{D^{est}}$
30: **if** $c_1 > 0$ **then**
31: **return** $[(c_1, round + 1), (\Delta_{D^{est}} - c_1, round)]$
32: **else if** $c_1 < 0$ **then**
33: **return** $[(-c_1, round - 1), (\Delta_{D^{est}} + c_1, round)]$
34: **end if**
35: **return** $[(\Delta_{D^{est}}, round)]$
36: **end function**

Δ_τ tuples and append the corresponding pairs to O_T^{est} (lines 10–13). Finally, it returns the estimated count-occurrence vector O_T^{est}.

The *estimated count-occurrence vectors* are computed by executing Algorithm 1 for each dataset. Let L_S and R_S be sample datasets of L and R, and O_{L_S} and O_{R_S} be the count-occurrence vectors derived after executing sampling queries with the respective GROUP BY clauses (see Section 4.2). Also, let $|L|_p^{est}$ and $|R|_p^{est}$ be the estimated number of tuples in L and R, respectively (satisfying the selection predicates). Note that $|L|_p^{est}$ and $|R|_p^{est}$ are computed during join cardinality estimation. Let $O_L^{est} = [(\delta_{l_1}^{est}, o_{l_1}^{est}), \ldots, (\delta_{l_x}^{est}, o_{l_x}^{est})]$ and $O_R^{est} = [(\delta_{r_1}^{est}, o_{r_1}^{est}), \ldots, (\delta_{r_y}^{est}, o_{r_y}^{est})]$ be the respective *estimated* count-occurrence vectors for L and R, where $D_L^{est} = \sum_{\forall x} \delta_{l_x}^{est}$ and $D_R^{est} = \sum_{\forall y} \delta_{r_y}^{est}$. We will use O_L^{est} and O_R^{est} to compute the *estimated frequency vectors* for both datasets as described next.

Since we maintain count-occurrence vectors as the common pairs (Section 4.2) and consider uniform distributions to estimate O_L^{est} and O_R^{est} (in the GetScaledUpVector and GetRestUniformVector functions), we have observed during the experiments that the number of pairs in these vectors are generally much smaller than D_L^{est} and D_R^{est}. As an example, compared to $D_L^{est} = 238510$ and $D_R^{est} = 5$, there are only four and three pairs in O_L^{est} and O_R^{est}, respectively (as shown in Table 2 for the running example).

5.3.2 Estimated Frequency Vector.

In equation (5), f_v corresponds to the fraction of tuples for a distinct value combination. Here, instead of the fraction of tuples for each distinct value combination, we compute the fraction of tuples for each $(\delta_{l_x}^{est}, o_{l_x}^{est}) \in O_L^{est}$ and $(\delta_{r_y}^{est}, o_{r_y}^{est}) \in O_R^{est}$ pairs and store them as the x^{th} and y^{th} elements, respectively, in the corresponding estimated frequency vectors.

Let $F_L^{est} = [\phi_1^l, \ldots, \phi_x^l]$ and $F_R^{est} = [\phi_1^r, \ldots, \phi_y^r]$ be the estimated frequency vectors for L and R, respectively. For the x^{th} pair $(\delta_{l_x}^{est}, o_{l_x}^{est}) \in O_L^{est}$, we compute the x^{th} fraction of tuples in F_L^{est} as $\phi_x^l = \frac{o_{l_x}^{est}}{|L|_p^{est}}$. Similarly, for the y^{th} pair $(\delta_{r_y}^{est}, o_{r_y}^{est}) \in O_R^{est}$, we compute the y^{th} fraction of tuples in F_R^{est} as $\phi_y^r = \frac{o_{r_y}^{est}}{|R|_p^{est}}$. Note that $\sum_{\forall x}(\phi_x^l \times \delta_{l_x}^{est}) = 1$ and $\sum_{\forall y}(\phi_y^r \times \delta_{r_y}^{est}) = 1$.

5.3.3 Estimated Distinct Cardinality.

The proposed MAMD approach uses the estimated frequency vectors F_L^{est} and F_R^{est}, the estimated count-occurrence vectors O_L^{est} and O_R^{est}, and the estimated join cardinality $N_{\bowtie}^{est}$, and estimates the number of distinct values D^{est} in a set of attributes in a 2-dataset join query by rewriting equation (5) as follows:

$$D^{est} = M - \sum_{\forall x} \sum_{\forall y} \left(\left(1 - \phi_x^l \times \phi_y^r\right)^{N_{\bowtie}^{est}} \times \delta_{l_x}^{est} \times \delta_{r_y}^{est} \right) \quad (6)$$

where,

- Number of possible distinct combinations, $M = D_L^{est} \times D_R^{est}$
- Estimated count-occurrence vectors,
  - $O_L^{est} = [(\delta_{l_1}^{est}, o_{l_1}^{est}), \ldots, (\delta_{l_x}^{est}, o_{l_x}^{est})]$
  - $O_R^{est} = [(\delta_{r_1}^{est}, o_{r_1}^{est}), \ldots, (\delta_{r_y}^{est}, o_{r_y}^{est})]$
- Estimated frequency vectors,
  - $F_L^{est} = [\phi_1^l, \ldots, \phi_x^l]$
  - $F_R^{est} = [\phi_1^r, \ldots, \phi_y^r]$
- Estimated join cardinality, $N_{\bowtie}^{est}$

Here, D^{est} is upper-bounded by the estimated join cardinality $N_{\bowtie}^{est}$. Also, as mentioned in Section 5.3.1, the number of pairs in the estimated vectors can be much smaller than D_L^{est} and D_R^{est}. When the exponent, i.e., $N_{\bowtie}^{est}$, is very large, the summation portion of the equation can result in a small number (like equation (5) [42]).

Note that the estimated join cardinality $N_{\bowtie}^{est}$ has a significant impact on equation (6). As a result, the effectiveness of MAMD depends on having a good join cardinality estimator. For example, suppose the original dataset is *highly skewed* and the predicates are *highly selective*. In such cases, if we consider a sample-based join cardinality estimator, the sample may not provide enough information to the join estimator, leading it to create bad estimates of $N_{\bowtie}^{est}$, eventually affecting the proposed MAMD approach.

Like other sample-based cardinality estimation methods, MAMD operates under two key assumptions:

- **Uniformity:** It assumes that values in each join attribute are *uniformly* distributed. This assumption is also used when computing the estimated count-occurrence vector (Section 5.3.1) and the estimated frequency vector (Section 5.3.2) for GROUP BY attributes. While this simplifies the estimation process, it may miss skewed patterns in real-world data.
- **Independence:** It assumes *independence* across attributes when estimating the number of distinct combinations (as in equation (5)), applying a basic probability principle that multiplies marginal probabilities. However, it ignores potential correlations between attributes, which can affect estimation accuracy.

If there is no selection predicate and the value combinations behave truly independently, the upper bound M might be sufficient due to these assumptions. However, in the presence of selection predicates in a *multi-dataset query*, the join cardinality $N_{\bowtie}^{est}$ bounds both lower (= $\min\{\max\{D_L^{est}, D_R^{est}\}, N_{\bowtie}^{est}\}$) and upper (= $\min\{D_L^{est} \times D_R^{est}, N_{\bowtie}^{est}\}$) bounds, and M cannot reflect the distribution of distinct value combinations satisfying the predicates. Incorporating the sample-derived frequency information, as done in equation (6), therefore provides a more stable estimate across queries with diverse predicate conditions.

We will revisit the impacts of these assumptions during our experimental evaluation using the real-world IMDB dataset and further discuss the limitations in Section 6.4.

The Same Running Example: Let us consider again the running example presented in Section 5.1 to estimate the distinct cardinality with the MAMD approach. Here, given that the left dataset is `LineItem`, $|L| = 6001215$, $n_p = 691$, $d_p = 690$, and $D_L^{est} = 238510$. Table 2 shows the returned values from the sampling queries and the corresponding estimated values required for equation (6).

Among 690 distinct values in the sample dataset L_S, 689 distinct values appear once and 1 distinct value appears twice, so the count-occurrence vector, $O_{L_S} = [(689, 1), (1, 2)]$. Since these 690 distinct values appear in L_S, the estimated number of tuples with these sampled distinct values becomes: $\tau_L = \left\lfloor \frac{690}{238510} \times 3901072 \right\rceil = 11286$, where the estimated number of tuples in `LineItem` satisfying the selection predicates $|L|_p^{est} = 3901072$.

We scale up the occurrences of these 690 sampled distinct values to obtain the corresponding estimated pairs $(689, 16)$ and $(1, 33)$. Then we apply the uniform distribution assumption to compute the estimated occurrences of the remaining 237820 distinct values (i.e., $\Delta_{D^{est}} = D_L^{est} - d_p = 238510 - 690$) over the remaining $\Delta_\tau = |L|_p^{est} - (689 \times 16 + 1 \times 33) = 3890015$ tuples. Here, $round = \left\lfloor \frac{3890015}{237820} \right\rceil = 16$, $c_1 = \Delta_\tau - round \times \Delta_{D^{est}} = 84895$, and $\Delta_{D^{est}} - c_1 = 152925$. Finally, the estimated *count-occurrence vector* of L becomes $O_L^{est} = [(689, 16), (1, 33), (84895, 17), (152925, 16)]$ while the corresponding estimated *frequency vector* becomes $F_L^{est} = [(4.10 \times 10^{-6}), (8.46 \times 10^{-6}), (4.36 \times 10^{-6}), (4.10 \times 10^{-6})]$.

Similarly, for the right dataset `Orders` (i.e., $|R| = 1500000$), $|R|_p^{est} = 53621$, and $D_R^{est} = 5$. Here, the estimated count-occurrence vector $O_R^{est} = [(1, 7055), (1, 16932), (3, 9878)]$ and the estimated frequency vector $F_R^{est} = [0.13157, 0.31577, 0.18422]$.

Since the estimated join cardinality is $N_{\bowtie}^{est} = 139455$ (satisfying the selection predicates), using equation (6), the estimated number of distinct values D^{est} becomes:

$$D^{est} = M - \sum_{\forall x} \sum_{\forall y} \left(\left(1 - \phi_x^l \times \phi_y^r\right)^{N_{\bowtie}^{est}} \times \delta_{l_x}^{est} \times \delta_{r_y}^{est} \right)$$
$$= 5 \times 238510 - 1061621 = 130929$$

Given that the actual distinct cardinality is $D^{act} = 134942$, the corresponding relative error $RE_p(\%)$ is:

$$RE_p(\%) = \frac{|134942 - 130929|}{144869} \times 100 = 2.77\%$$

Here, the relative error $RE_p(\%)$ drops to 2.77%, compared to 89% from the naive approach, demonstrating a substantial improvement by MAMD over the naive estimate.

Note that in this example, in the presence of selection predicates, $N_{\bowtie}^{est} < \max\{D_L^{est}, D_R^{est}\}$, and this results in equal lower and upper bounds of D^{est}, i.e., $\min\{\max\{238510, 5\}, N_{\bowtie}^{est}\} = 139445$ and $M = \min\{238510 \times 5, 139445\} = 139445$, respectively. This bound produces a relative error of 3.12%, compared to 2.77% from MAMD which incorporates sample-derived frequency information.

5.4 MAMD for *m*-dataset Joins

We can now proceed with MAMD for m-dataset joins. Since *left-deep trees* are commonly used during query optimization, we focus our description considering a left-deep query tree Q with m-datasets. However, our approach is also applicable to other query-tree types.

Algorithm 2 The MAMD Algorithm for m-dataset Joins

Input:
$\{T1, T2, \ldots, Tm\}$: m datasets in the query tree Q
$J = [\bowtie_1, \bowtie_2, \ldots, \bowtie_{m-1}]$: set of join nodes in Q
Output: estimated distinct cardinality D^{est}

```
GetJoinEstimates(J)
ComputeEstimatesInBaseDatasets({T1, T2, ..., Tm})
D^est ← 0, O^est ← []
for each (L ⋈_j R) ∈ J do                    ▷ iterate j from 1 to m − 1
    if L is not a base dataset then
        D_L^est ← D^est, O_L^est ← O^est      ▷ values from the last join
    end if
    if D_L^est = 0 then
        D^est ← D_R^est, O^est ← O_R^est
    else if D_R^est = 0 then
        D^est ← D_L^est, O^est ← O_L^est
    else
        sum ← 0
        for each (δ_lx^est, o_lx^est) ∈ O_L^est, φ_x^l ∈ F_L^est do
            for each (δ_ry^est, o_ry^est) ∈ O_R^est, φ_y^r ∈ F_R^est do
                sum ← sum + (1 − φ_x^l × φ_y^r)^N_⋈j^est × (δ_lx^est × δ_ry^est)
            end for
        end for
        D^est ← D_L^est × D_R^est − sum
        O^est ← GetRestUniformVector(D^est, N_⋈j^est)
    end if
end for
return D^est
```

Table 2: Obtained values for the MAMD approach for the running example

Dataset	n_p	d_p	Est. dataset cardinality, N_T^{est}	Est. distinct cardinality, D_T^{est}	Count-occurrence vector pairs (δ_{t_k}, o_{t_k}) from samples	Est. count-occurrence vector pairs, $(\delta_{t_k}^{est}, o_{t_k}^{est})$	Est. frequency vector elements, $\phi_k^t = o_{t_k}^{est}/N_T^{est}$
LineItem	691	690	3901072	238510	(689, 1)	(689, 16)	4.10×10^{-6}
					(1, 2)	(1, 33)	8.46×10^{-6}
						(84895, 17)	4.36×10^{-6}
						(152925, 16)	4.10×10^{-6}
Orders	38	5	53621	5	(1, 5)	(1, 7055)	0.13157
					(1, 12)	(1, 16932)	0.31577
					(3, 7)	(3, 9878)	0.18422

While in real life, the query tree changes during query optimization (by shuffling the join orders), for simplicity of presentation, we assume that the query tree (not necessarily optimal) is known and does not change. Let Q be a left-deep tree with m-datasets $\{T1, T2, \ldots, Tm\}$ and $J = [\bowtie_1, \bowtie_2, \ldots, \bowtie_{m-1}]$ be a set of corresponding join nodes in Q. Here, $1 \leq j \leq (m-1)$ in the subscript of a join node $\bowtie_j$ defines the level of the node in Q and $\bowtie_{m-1}$ is the highest join node in Q.

Algorithm 2 describes the MAMD approach for m-dataset joins. It takes Q and J as inputs and returns the estimated distinct cardinality D^{est} of the highest join node of Q as the output. First, it calls the GetJoinEstimates procedure to collect the estimated join cardinality $N_{\bowtie_j}^{est}$ values for join nodes $\bowtie_j \in J$, satisfying the respective selection predicates. Note that these $N_{\bowtie_j}^{est}$ values are computed during join estimation. Next, the ComputeEstimatesInBaseDatasets procedure is executed to compute the required estimated values for each base dataset Ti, where $1 \leq i \leq m$. This procedure computes the estimated distinct cardinality D_{Ti}^{est} using equation (2), the estimated count-occurrence vector O_{Ti}^{est} by executing the GetEstimatedVector procedure (Algorithm 1), and the estimated frequency vector F_{Ti}^{est} from O_{Ti}^{est}, for each Ti dataset (Section 5.3.2).

Next, lines 13–19 evaluate equation (6) to estimate D^{est} for each $\bowtie_j \in J$, where L and R are the left and right child nodes of $\bowtie_j$. Also, to compute the estimated count-occurrence vector O^{est}, the corresponding estimated number of distinct values D^{est} are distributed uniformly over the estimated number of tuples $N_{\bowtie_j}^{est}$ of the join node $\bowtie_j$ by executing the GetRestUniformVector function (line 20). If L is a join node, we propagate the last computed D^{est} and O^{est} values to the left child node (lines 5–7). Also, if there are no attributes in the distinct combination from L (or R), we propagate D_R^{est} and O_R^{est} (or, D_L^{est} and O_L^{est}) values to the join node and continue to the next level (lines 8–11). Finally, D^{est} for the highest join node $\bowtie_{m-1}$ is returned as the output.

Generality of the MAMD Approach: If Q is not a *left-deep tree*, lines 5–7 of Algorithm 2 need to be modified: (1) if Q is a *right-deep tree* or a *zig-zag tree*, we propagate the D^{est} and O^{est} values of the j^{th} join node to the corresponding child node of the $(j+1)^{\text{th}}$ join node, while iterating j from 1 to $m-2$, and (2) if Q is a *bushy tree*, we compute the D^{est} and O^{est} values and store them with the corresponding join node so that we can use them for the respective higher-level join node during the bottom-up iteration of nodes in Q. The rest of the steps are the same as in Algorithm 2.

6 EXPERIMENTAL EVALUATION

We proceed with the experimental evaluation of the proposed *MAMD* (Multi-Attribute, Multi-Dataset) approach for distinct cardinality estimation from two primary perspectives: (1) the compile-time overhead introduced by the estimation algorithms, and (2) the average relative error (%) as defined in Section 5. We further assess the approach's estimation accuracy, efficiency, and scalability across varying dataset sizes and sample sizes.

System Configuration: All experiments were carried out using Apache AsterixDB [1], deployed on a single Amazon EC2 instance. The instance was configured with 8 vCPUs (Intel(R) Xeon(R) Platinum 8375C CPU @ 2.90 GHz), 32 GB of RAM, and 2 TB of SSD storage, running 64-bit Ubuntu 22.04 LTS. For the experiments, we integrated MAMD into the cost-based optimizer (CBO) of AsterixDB to evaluate distinct cardinality estimates during query optimization. This evaluation is done without GROUP BY pushdown or operator rearrangement in query plans. All count values use 64-bit integers; distinct and join cardinalities use 64-bit floats with two decimal digits, and fractions use 10-decimal precision unless they are explicitly rounded to the nearest integer.

6.1 Datasets and Samples

To evaluate the performance of the proposed sample-based *MAMD* approach for estimating the number of distinct values in a *set of attributes* in a *multi-dataset* query, we used both synthetic and real-world datasets.

We first considered the TPC-H [34] benchmark and generated TPC-H datasets at scale factors (SF) of 1, 10, 30, and 100. Each SF corresponds to the generated database size, e.g., SF = 1 and SF = 30 refer to 1 GB and 30 GB, respectively. Each dataset, except `Nation` and `Region`, grows linearly with the SF. At SF = 30, the largest datasets, `LineItem` and `Orders`, contain approximately 180 million and 45 million tuples, respectively. The data is synthetically generated and uniformly distributed.

For real-world evaluation, we used the IMDB datasets as prepared for the JOB [28] benchmark. In contrast to TPC-H, IMDB contains attribute correlations, non-uniform data distributions, and many-to-many relationships. There are 21 datasets, of which the largest two datasets, `cast_info` and `movie_info`, have 36 million and 15 million tuples, respectively. The full database size is ~ 3.6 GB.

All datasets were loaded into Apache AsterixDB, which supports sampling via its ANALYZE statement. AsterixDB can collect

and store a *fixed-size* sample per dataset, regardless of the underlying dataset size. Sampling is performed without replacement and distributed across partitions to minimize storage and network overhead. If a sample dataset already exists for a given dataset, the ANALYZE statement replaces the previously *stored sample*, ensuring that only one sample is maintained per dataset at a time.

In AsterixDB, samples are collected and stored ahead of time, rather than being generated dynamically during query compilation. In our experiments, we varied the sample size for all datasets using three system-defined configurations:

(1) **Low:** 1063 tuples per dataset
(2) **Medium:** 4× the low size (4252 tuples)
(3) **High:** 16× the low size (17008 tuples)

For each experimental configuration, the corresponding sample datasets were used by the MAMD algorithms during query compilation to estimate distinct cardinalities.

6.2 Evaluation using TPC-H Datasets

6.2.1 TPC-H Query Workload. While we use the original TPC-H datasets for evaluation, we do not use the benchmark's standard set of 22 queries. Those queries are primarily designed to evaluate the performance of decision support systems and rarely involve GROUP BY clauses across multiple attributes and datasets, which are essential for assessing distinct cardinality estimation for multiple attributes in join queries. To address this, we define a custom set of queries for our TPC-H workload over the TPC-H schema.

Our TPC-H query workload is designed to evaluate the performance of the proposed MAMD approach across queries with varying join complexities, using TPC-H datasets at SF = 1, 10, 30, and 100. These queries involve *primary-foreign key* joins and contain selection predicates, including equality, range, IN, and/or LIKE predicates. Each query includes COUNT and GROUP BY aggregates, and 2-4 attributes (from at least two datasets) in the GROUP BY clause. Based on the number of datasets involved in the joins, the workload is divided into five query sets, each containing a different number of queries, as summarized in Table 3, resulting in a total of 58 queries. The complete set of queries used in our TPC-H workload is publicly available [1].

Table 3: Summary of the TPC-H Query Workload

Query set	No. of queries	No. of Group-By attributes
2-datasets	15	2-3
3-datasets	15	2-3
4-datasets	10	3-4
5-datasets	9	3-4
6-datasets	9	2-3

6.2.2 Query Performance and Relative Error (%). We first measured the query compilation time, which reflects the total time taken by CBO of AsterixDB to generate an optimal execution plan, including but not limited to the overhead introduced by the MAMD estimation algorithms. Figure 2 reports the average query compilation and execution times (averaged over the 58 queries in our TPC-H workload) as SF increases from 1 to 100. We observe that query compilation time (Figures 2b-2c) remains relatively stable across varying sample sizes and scale factors. However, higher scale factors can lead to more distinct values in attributes with larger domains. Eventually, this can increase the number of entries in the estimated count-occurrence and frequency vectors, and thus the computational complexity of equation (6). As a result, we observe a small relative difference in compilation time, ranging from 0.01–0.1 seconds, as SF increases. The overall results indicate that the integration of sample-based approaches (Naive and MAMD) into CBO adds minimal computational overhead. In contrast, query execution time (Figure 2d) increases with larger datasets, as expected, due to higher I/O and join processing costs. Also, we can observe that both compilation and execution times with these sample-based approaches are similar to those without samples (Figure 2a).

We selected the "high" sample configuration for further analysis, since it consistently produced more accurate estimates while keeping compilation time nearly unchanged (within 0.02 to 0.07 seconds of the other settings). Although it still represents only a small fraction of the data (0.28% to 0.00028% across SF), it is the only system-defined configuration large enough (ideally 0.1%-1%) to provide reliable cardinality estimates [15, 17]. Figure 3 shows the average relative error, $RE_p(\%)$, over the TPC-H query workload at scale factors ranging from 1 to 100. For this set of experiments, we report the average $RE_p(\%)$ for each query set using the "high" sample size, where join queries are grouped based on the number of datasets participating in joins. Since both N_p^{act} and D^{act} vary based on join attributes and selection predicates, and do not change proportionately with the number of datasets, we observe that the average $RE_p(\%)$ varies among groups. Additionally, the results show that MAMD (Figure 3b) maintains low relative error percentages, ranging from $0\% - 3.5\%$ across all query groups and scale factors, while the naive approach (Figure 3a) exhibits much higher errors ($0.05\% - 27.02\%$ on average, with some queries reaching 98-99%).

Since TPC-H data is generated with *uniform distributions*, which aligns with MAMD's uniformity assumption, these results reflect an ideal scenario. This demonstrates that MAMD can deliver high-quality estimates with low compilation overhead and scales effectively with increasing dataset sizes. However, real-world datasets often exhibit correlations and non-uniform data distributions, which can impact estimation accuracy and increase $RE_p(\%)$. These factors highlight important limitations of the proposed MAMD approach, as we will explore in the next section using the IMDB datasets.

6.3 Evaluation using IMDB Datasets

For this set of experiments, we use the IMDB datasets from May 2013, originally exported to CSV format for the Join Order Benchmark (JOB) [28], and adopt the Cardinality Estimation Benchmark (CEB) [31] queries built on top of these datasets.

6.3.1 CEB-IMDB Query Workload. Though JOB consists of join queries with various selection predicates, it was not designed to evaluate distinct cardinality estimation. In contrast, CEB contains several join queries with GROUP BY clauses and diverse selection predicates, making it better suited for evaluating our approach. The CEB queries are based on the IMDB schema and include both *primary-foreign key* and *foreign-foreign key* joins. Originally designed to evaluate supervised learning-based cardinality estimation methods, CEB includes a total of 13644 IMDB queries across 16

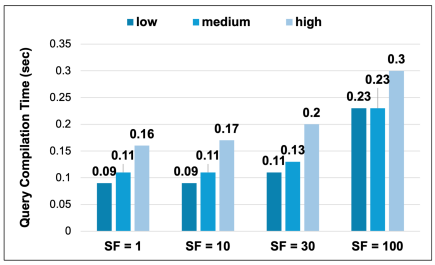
[1] https://github.com/MehnazMahin/Distinct_Card_TPCH

SF	Query compilation time	Query execution time
1	0.11	1.40
10	0.11	14.06
30	0.20	46.66
100	0.23	148.23

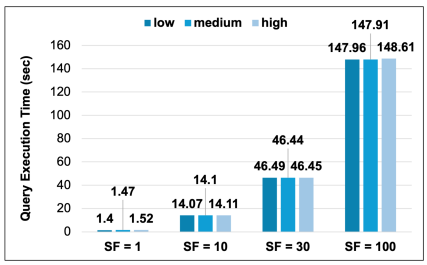
(a) Without samples

SF	Query compilation time			Query execution time		
	low	medium	high	low	medium	high
1	0.10	0.11	0.17	1.38	1.38	1.41
10	0.10	0.12	0.17	13.64	13.24	13.3
30	0.18	0.18	0.22	45.52	43.84	43.83
100	0.23	0.23	0.31	145.12	145.65	145.68

(b) Naive

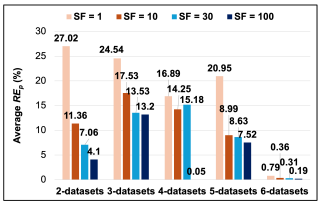


(c) Query compilation time using MAMD

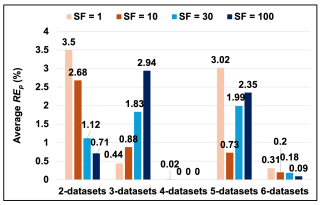


(d) Query execution time using MAMD

Figure 2: Query compilation and execution times (measured in seconds) using TPC-H datasets at SF = 1, 10, 30, and 100.



(a) Naive



(b) MAMD

Figure 3: Average $RE_p(\%)$ over the TPC-H query workload using the "high" sample size at SF = 1, 10, 30, and 100.

query templates [2]. Among them, six templates contain queries with GROUP BY clauses: 3b, 9a, 9b, 10a, 11a, and 11b.

Queries from these templates contain IN predicates on the `kind` attribute of the `kind_type` dataset and the `role` attribute of the `role_type` dataset, using static lists of string values. The exception is 11a, where the IN predicate appears only on `role`. Upon careful analysis, we found that several of the GROUP BY attributes – particularly from the `title` and `person_info` datasets – are highly correlated with `role` and `kind`, and that the overall data distribution of the IMDB datasets is skewed. When any string values in these IN-lists are filtered out by other selection predicates, it becomes challenging to detect such effects from *stored samples* at query compilation time. As a result, estimating the number of distinct value combinations across joins becomes difficult.

Due to this limitation, we excluded templates 3b and 9a from our workload, as they frequently exhibit this issue. In these templates, selection predicates reduce join cardinalities to very small values (typically 1–100 and always <13000), leaving distinct cardinalities nearly identical to the join ones, whereas the same joins without predicates produce 3–4 billion tuples. This extreme reduction leaves too few overlapping join keys for meaningful distinct-estimation behavior. To evaluate the performance of the proposed sample-based MAMD approach, we selected 10 random queries from each of the remaining four templates – 9b, 10a, 11a, and 11b – resulting in a total of 40 queries in our CEB-IMDB query workload. Since queries in each template share the same join structure and group-by attributes and *differ only in predicate values*, these 40 randomly selected queries are sufficient to evaluate the performance of MAMD on real-world datasets. Table 4 presents a summary of the selected templates and their characteristics for the CEB-IMDB workload.

[2]CEB queries can be found: https://github.com/RyanMarcus/imdb_pg_dataset/

Table 4: Summary of the CEB-IMDB Query Workload

Query template	No. of datasets	No. of Group-By attributes
9b	9	2
10a	7	2
11a	10	3
11b	12	3

6.3.2 Query Performance and Relative Error (%). As we did for the experiments using TPC-H datasets, we evaluated the query performance in terms of query compilation and execution times for the CEB-IMDB workload. Figure 4 reports the average query compilation and execution times (averaged over 10 queries of each template in the CEB-IMDB workload). Similar to the results observed for the TPC-H workload, compilation time remains relatively stable across the workload, with minimal difference in compilation time among sample sizes, typically within 0.01 to 0.1 seconds (Figures 4b-4c). We can observe that query compilation time increases with the number of datasets involved in a query template, resulting in lower compilation time for '10a' using both the naive and MAMD approaches. In contrast, query execution time (Figure 4d) varies more significantly due to the complexity and size of the IMDB datasets, as expected. Note that similar to the TPC-H workload, both compilation and execution times with these sample-based approaches remain consistent with the times without samples (Figure 4a).

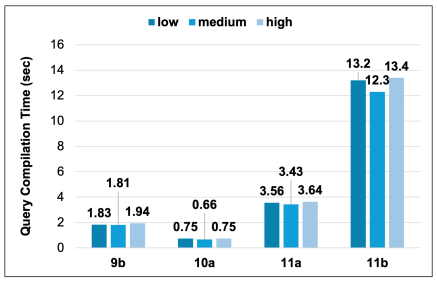
As in the previous evaluation (Section 6.2.2), we continue with the "high" sample configuration for relative error (%) analysis, since it consistently provides more accurate results with negligible compile-time impact. Figure 5 presents the average relative error, $RE_p(\%)$, for each of the four query templates in the CEB-IMDB workload. Here, the join queries are grouped by their respective templates, and results are reported using the "high" sample size (e.g., "high" sample represents 0.12% of `movie_info`). We observe that the average relative error ranges from 1.6% – 31% using MAMD, while ranging from 2% – 45% using the naive approach, depending on the complexity of the join predicates and selection predicates in the template. These error rates are notably higher than those observed in the TPC-H workload. This increase is expected, as the IMDB datasets exhibit non-uniform data distributions and strong attribute-level correlations, which conflict with MAMD's uniformity and independence assumptions. Note that these higher error rates are not specific to MAMD. Any sample-based estimator, including the naive approach, is fundamentally limited when the underlying data exhibits strong correlations or skewness.

Set	Query compilation time	Query execution time
9b	1.75	37.03
10a	0.63	38.13
11a	3.31	46.11
11b	12.82	55.13

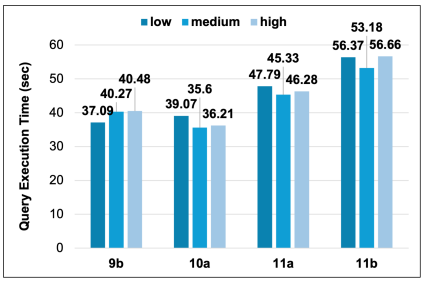
(a) Without samples

Set	Query compilation time			Query execution time		
	low	medium	high	low	medium	high
9b	1.71	1.72	1.71	36.06	37.78	38.53
10a	0.61	0.57	0.63	36.25	40.55	40.64
11a	3.27	3.24	3.25	42.08	42.14	42.09
11b	12.50	12.38	12.21	54.29	54.45	54.45

(b) Naive

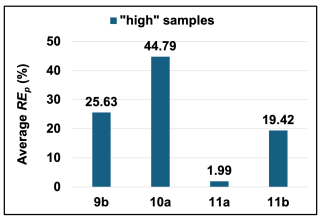


(c) Query compilation time using MAMD

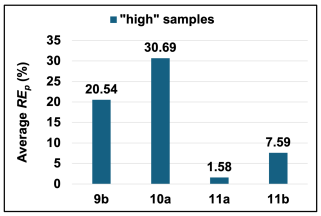


(d) Query execution time using MAMD

Figure 4: Query compilation and execution times (measured in seconds) using IMDB datasets and the CEB-IMDB query workload.



(a) Naive



(b) MAMD

Figure 5: Average $RE_p(\%)$ over the CEB-IMDB query workload (with the "high" sample size, grouped by query templates).

Despite the relatively high $RE_p(\%)$ in certain cases, the distinct estimates remain sufficiently effective for guiding optimizer decisions, such as choosing between hash-based and sort-based GROUP BY algorithms, one of the key applications of distinct cardinality estimation. However, the impact of the skewed data distributions and attribute correlations in the IMDB datasets points to some limitations of the MAMD approach.

6.4 Limitations of the MAMD Approach

Similar to traditional sample-based cardinality estimators [7, 9, 28], MAMD relies on *uniformity* and *independence* assumptions (as discussed in Section 5.3.3). Thus, it cannot capture attribute-level correlations, particularly those that span multiple datasets involved in joins or that arise across multiple-dataset attributes in GROUP BY clauses. The uniformity assumption can also affect the major components of MAMD – the estimated count-occurrence (Section 5.3.1) and estimated frequency (Section 5.3.2) vectors.

However, in real-world scenarios, attribute values may not be uniformly distributed (perhaps even *highly skewed*) and/or attribute value combinations can be *highly correlated* across multiple datasets. As we discussed in Section 6.3.1, some attributes in IMDB's datasets such as `cast_info`, `person_info`, `title`, `role_type`, etc., are highly correlated. Capturing such correlations or data skewness is inherently challenging for any sample-based estimators, including the naive and MAMD approaches, and ignoring these dependencies can significantly degrade estimation accuracy. In some such cases, the simple heuristic employed in commercial DBMS optimizers for estimating distinct cardinalities, i.e., the multiplication of individual distinct counts, can outperform MAMD.

Moreover, the performance of MAMD depends on having representative samples and a good join cardinality estimator. If the sample size is too small or cannot adequately represent the data distributions, the performance of sample-based join estimators will degrade, eventually affecting the performance of MAMD.

7 CONCLUSION

In this paper, we addressed one of the classical and open problems of cost-based optimizers: estimating the number of distinct values in an attribute or a set of attributes in *multi-dataset* queries – also known as the *distinct cardinality estimation* problem. It becomes challenging in the presence of selection predicates, especially without exploring a substantial fraction of the datasets. We proposed MAMD (Multi-Attribute, Multi-Dataset), a sample-based approach for estimating distinct values in a set of attributes (or an attribute) in multi-dataset queries, with or without selection predicates. It is also applicable to single-dataset queries. The proposed MAMD approach is designed to operate using stored samples instead of examining actual datasets, making it well-suited for large-scale datasets where computing accurate distinct cardinalities is not feasible. Our experiments using both synthetic and real-world datasets, namely the TPC-H and the IMDB benchmark datasets, demonstrate that MAMD estimates distinct cardinalities with moderately low relative error, while maintaining low storage and computational overhead. It also exhibits robustness as the dataset size scales.

While MAMD offers a practical and scalable solution for distinct cardinality estimation during query optimization, it assumes uniformity in join attribute values and does not account for attribute-level correlations across multiple datasets. Consequently, MAMD does not do well when the results of join queries exhibit significant correlations. Future work may include relaxing these assumptions and extending the proposed sample-based approach by incorporating correlation-aware sampling techniques and non-uniformity adjustments. Future work could also consider adapting queries from other benchmarks, such as DSB [11], to assess MAMD's performance.

ACKNOWLEDGEMENTS

We thank Murali Krishna and Vijay Sarathy from Couchbase, Inc. for input on the sample-based cost-based optimizer implementation in Apache AsterixDB. Their insights contributed to the initial exploration of our work. This research was supported in part by NSF awards CNS-1925610, CNS-1924694, IIS-1954644, NIFA award 2024-67022-43695, an industrial gift from Google, and funding from the Donald Bren Foundation (via a Bren Chair at UC Irvine).

REFERENCES

[1] 2024. Apache AsterixDB. Retrieved Novermber 10, 2024 from https://asterixdb.apache.org/

[2] 2024. Cardinality Estimation (SQL Server). Retrieved Novermber 22, 2024 from https://learn.microsoft.com/en-us/sql/relational-databases/performance/cardinality-estimation-sql-server?view=sql-server-ver16

[3] 2024. PostgreSQL Documentation 17: Chapter 50.5. Planner/Optimizer. Retrieved Novermber 21, 2024 from https://www.postgresql.org/docs/current/planner-optimizer.html

[4] Swarup Acharya, Phillip B. Gibbons, and Viswanath Poosala. 2000. Congressional Samples for Approximate Answering of Group-By Queries. *SIGMOD Rec.* 29, 2 (2000), 487–498.

[5] K.O. Bowman and L. R. Shenton. 1985. Method of moments. *Encyclopedia of Statistical Sciences* 5 (1985), 467–453.

[6] J. Bunge and M. Fitzpatrick. 1993. Estimating the Number of Species: A Review. *J. Amer. Statist. Assoc.* 88, 421 (1993), 364–373.

[7] Moses Charikar, Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. 2000. Towards Estimation Error Guarantees for Distinct Values. In *PODS Symposium*. 268–279.

[8] Surajit Chaudhuri, Rajeev Motwani, and Vivek Narasayya. 1999. On Random Sampling over Joins. *SIGMOD Rec.* 28, 2 (1999), 263–274.

[9] Graham Cormode, Minos Garofalakis, Peter J Haas, and Chris Jermaine. 2012. Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches. *Foundations and Trends in Databases* 4, 1-3 (2012), 1–294.

[10] Shiyuan Deng, Shangqi Lu, and Yufei Tao. 2023. On Join Sampling and the Hardness of Combinatorial Output-Sensitive Join Algorithms. In *SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 99–111.

[11] Bailu Ding, Surajit Chaudhuri, Johannes Gehrke, and Vivek Narasayya. 2021. DSB: a decision support benchmark for workload-driven and traditional database systems. *VLDB* 14, 13 (2021), 3376–3388.

[12] Anshuman Dutt, Chi Wang, Azade Nazi, Srikanth Kandula, Vivek Narasayya, and Surajit Chaudhuri. 2019. Selectivity Estimation for Range Predicates using Lightweight Models. *Proc. VLDB Endow.* 12, 9 (2019), 1044–1057.

[13] P. Flajolet, Éric Fusy, O. Gandouet, and F. Meunier. 2007. HyperLogLog: The analysis of a near-optimal cardinality estimation algorithm. In *Analysis of Algorithms*. 127–146.

[14] Michael J. Freitag and Thomas Neumann. 2019. Every Row Counts: Combining Sketches and Sampling for Accurate Group-By Result Estimates. In *Conference on Innovative Data Systems Research*.

[15] Phillip B. Gibbons. 2001. Distinct Sampling for Highly-Accurate Answers to Distinct Values Queries and Event Reports. In *VLDB*. 541–550.

[16] Dimitrios Gunopulos, George Kollios, Vassilis J. Tsotras, and Carlotta Domeniconi. 2005. Selectivity estimators for multidimensional range queries over real attributes. *VLDB* 14, 2 (2005), 137–154.

[17] Peter J. Haas, Jeffrey F. Naughton, S. Seshadri, and Lynne Stokes. 1995. Sampling-Based Estimation of the Number of Distinct Values of an Attribute. In *VLDB*. 311–322.

[18] Peter J. Haas, Jeffrey F. Naughton, S. Seshadri, and Arun N. Swami. 1996. Selectivity and Cost Estimation for Joins Based on Random Sampling. *J. Comput. System Sci.* 52, 3 (1996), 550–569.

[19] Hazar Harmouch and Felix Naumann. 2017. Cardinality Estimation: An Experimental Survey. *Proc. VLDB Endow.* 11, 4 (2017), 499–512.

[20] Roman Heinrich, Manisha Luthra, Johannes Wehrstein, Harald Kornmayer, and Carsten Binnig. 2025. How Good are Learned Cost Models, Really? Insights from Query Optimization Tasks. *Proc. ACM Manag. Data* 3, 3, Article 172 (2025).

[21] Joseph M. Hellerstein, Peter J. Haas, and Helen J. Wang. 1997. Online Aggregation. In *ACM SIGMOD*. 171–182.

[22] Stefan Heule, Marc Nunkesser, and Alex Hall. 2013. HyperLogLog in Practice: Algorithmic Engineering of a State of The Art Cardinality Estimation Algorithm. In *EDBT*. 683–692.

[23] Wen-Chi Hou, Gultekin Ozsoyoglu, and Baldeo K. Taneja. 1988. Statistical Estimators for Relational Algebra Expressions. In *PODS Symposium*. 276–287.

[24] Wen-Chi Hou, Gultekin Ozsoyoglu, and Baldeo K. Taneja. 1989. Processing Aggregate Relational Queries with Hard Time Constraints. *SIGMOD Rec.* 18, 2 (1989), 68–77.

[25] Yannis E. Ioannidis and Stavros Christodoulakis. 1991. On the Propagation of Errors in the Size of Join Results. *SIGMOD Rec.* 20, 2 (1991), 268–277.

[26] Andreas Kipf, Michael J. Freitag, Dimitri Vorona, Peter Boncz, Thomas Neumann, and Alfons Kemper. 2019. Estimating Filtered Group-By Queries is Hard: Deep Learning to the Rescue. In *1st International Workshop on Applied AI for Database Systems and Applications*.

[27] Donald E. Knuth. 1997. *The Art of Computer Programming, Volume 2 (3rd ed.): Seminumerical Algorithms* (3 ed.). Addison-Wesley.

[28] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How Good Are Query Optimizers, Really? *Proc. VLDB Endow.* 9, 3 (2015), 204–215.

[29] Runzhong Li, Qilong Li, Haotian Liu, Rui Mao, Qing Li, and Bo Tang. 2025. Athena: An Effective Learning-based Framework for Query Optimizer Performance Improvement. *Proc. ACM Manag. Data* 3, 3, Article 132 (2025).

[30] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2022. Bao: Making Learned Query Optimization Practical. *SIGMOD Rec.* 51, 1 (2022), 6–13.

[31] Parimarjan Negi, Ryan Marcus, Andreas Kipf, Hongzi Mao, Nesime Tatbul, Tim Kraska, and Mohammad Alizadeh. 2021. Flow-Loss: Learning Cardinality Estimates That Matter. *Proc. VLDB Endow.* 14, 11 (2021), 2019–2032.

[32] Trong Duc Nguyen, Ming-Hung Shih, Sai Sree Parvathaneni, Bojian Xu, Divesh Srivastava, and Srikanta Tirthapura. 2019. Random Sampling for Group-By Queries. In *ICDE*. 541–552.

[33] G. Ozsoyoglu, K. Du, A. Tjahjana, WC. Hou, and D.Y. Rowland. 1991. On Estimating COUNT, SUM, and AVERAGE Relational Algebra Queries. In *Database and Expert Systems Applications*. 406–412.

[34] Meikel Poess and Chris Floyd. 2000. New TPC benchmarks for decision support and web commerce. *SIGMOD Rec.* 29, 4 (2000), 64–71.

[35] Viswanath Poosala, Peter J. Haas, Yannis E. Ioannidis, and Eugene J. Shekita. 1996. Improved Histograms for Selectivity Estimation of Range Predicates. *SIGMOD Rec.* 25, 2 (1996), 294–305.

[36] Silvan Reiner and Michael Grossniklaus. 2023. Sample-Efficient Cardinality Estimation Using Geometric Deep Learning. *VLDB* 17, 4 (2023), 740–752.

[37] P. Griffiths Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. 1979. Access Path Selection in a Relational Database Management System. In *SIGMOD*. 23–34.

[38] A. Shlosser. 1981. On estimation of the size of the dictionary of a long text on the basis of a sample. *Engineering Cybernetics* 19 (1981), 97–102.

[39] Xiaoying Wang, Changbo Qu, Weiyuan Wu, Jiannan Wang, and Qingqing Zhou. 2021. Are we ready for learned cardinality estimation? *Proc. VLDB Endow.* 14, 9 (2021), 1640–1654.

[40] Zongheng Yang, Wei-Lin Chiang, Sifei Luan, Gautam Mittal, Michael Luo, and Ion Stoica. 2022. Balsa: Learning a Query Optimizer Without Expert Demonstrations. In *SIGMOD*. 931–944.

[41] Xiang Yu, Chengliang Chai, Guoliang Li, and Jiabin Liu. 2022. Cost-Based or Learning-Based? A Hybrid Query Optimizer for Query Plan Selection. *VLDB* 15, 13 (2022), 3924–3936.

[42] Xiaohui Yu, Calisto Zuzarte, and Kenneth C. Sevcik. 2005. Towards Estimating the Number of Distinct Value Combinations for a Set of Attributes. In *CIKM*. 656–663.

[43] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. 2018. Random Sampling over Joins Revisited. In *SIGMOD*. 1525–1539.