Extensible Query Optimizers in Practice

Other titles in Foundations and Trends® in Databases

Modern Techniques For Querying Graph-structured Databases
Amine Mhedhbi, Amol Deshpande and Semih Salihoğlu
ISBN: 978-1-63828-424-6

More Modern B-Tree Techniques
Goetz Graefe
ISBN: 978-1-63828-372-0

Consensus in Data Management: From Distributed Commit to Blockchain
Faisal Nawab and Mohammad Sadoghi
ISBN: 978-1-63828-160-3

Multidimensional Array Data Management
Florin Rusu
ISBN: 978-1-63828-148-1

Modern Datalog Engines
Bas Ketsman and Paraschos Koutris
ISBN: 978-1-63828-042-2

Natural Language Interfaces to Data
Abdul Quamar, Vasilis Efthymiou, Chuan Lei and Fatma Özcan
ISBN: 978-1-63828-028-6

Extensible Query Optimizers in Practice

Bailu Ding
Microsoft Corporation
badin@microsoft.com

Vivek Narasayya
Microsoft Corporation
viveknar@microsoft.com

Surajit Chaudhuri
Microsoft Corporation
surajitc@microsoft.com

now
the essence of knowledge
Boston — Delft

Foundations and Trends® in Databases

Published, sold and distributed by:
now Publishers Inc.
PO Box 1024
Hanover, MA 02339
United States
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

Outside North America:
now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

The preferred citation for this publication is

B. Ding *et al.*. *Extensible Query Optimizers in Practice*. Foundations and Trends® in Databases, vol. 14, no. 3-4, pp. 186–402, 2024.

ISBN: 978-1-63828-453-6

Foundations and Trends® in Databases
Volume 14, Issue 3-4, 2024

Editorial Board

Editorial Scope

Foundations and Trends® in Databases publishes survey and tutorial articles in the following topics:

- Data Models and Query Languages
- Query Processing and Optimization
- Storage, Access Methods, and Indexing
- Transaction Management, Concurrency Control and Recovery
- Deductive Databases
- Parallel and Distributed Database Systems
- Database Design and Tuning
- Metadata Management
- Object Management
- Trigger Processing and Active Databases
- Data Mining and OLAP
- Approximate and Interactive Query Processing
- Data Warehousing
- Adaptive Query Processing
- Data Stream Management
- Search and Query Integration
- XML and Semi-Structured Data
- Web Services and Middleware
- Data Integration and Exchange
- Private and Secure Data Management
- Peer-to-Peer, Sensornet and Mobile Data Management
- Scientific and Spatial Data Management
- Data Brokering and Publish/Subscribe
- Data Cleaning and Information Extraction
- Probabilistic Data Management

Information for Librarians

Foundations and Trends® in Databases, 2024, Volume 14, 4 issues. ISSN paper version 1931-7883. ISSN online version 1931-7891. Also available as a combined paper and online subscription.

Contents

Extensible Query Optimizers in Practice

Bailu Ding, Vivek Narasayya and Surajit Chaudhuri

Microsoft Corporation, USA; badin@microsoft.com, viveknar@microsoft.com, surajitc@microsoft.com

ABSTRACT

The performance of a query crucially depends on the ability of the query optimizer to choose a good execution plan from a large space of alternatives. With the discovery of algebraic transformation rules and the emergence of new application-specific contexts, extensibility has become a key requirement for query optimizers. This monograph describes extensible query optimizers in detail, focusing on the Volcano/Cascades framework used by several database systems including Microsoft SQL Server. We explain the need for extensible query optimizer architectures and how the optimizer navigates the search space efficiently. We then discuss several important transformations that are commonly used in practice. We describe cost estimation, an essential component that the optimizer relies upon to quantitatively compare alternative plans in the search space. We discuss how database systems manage plans over their lifetime as data and workloads change. We conclude with a few open challenges.

Bailu Ding, Vivek Narasayya and Surajit Chaudhuri (2024), "Extensible Query Optimizers in Practice", Foundations and Trends® in Databases: Vol. 14, No. 3-4, pp 186–402. DOI: 10.1561/1900000077.

1

Introduction

SQL [134] is a high-level declarative language for querying relational data. It is the de-facto standard query language for relational data and is supported by all major relational database management systems (RDBMSs) and increasingly also by the Big Data Systems. SQL allows declarative specification of queries over relational data involving selections, joins, group-by, aggregation, and nested sub-queries, which are important for a wide variety of decision support queries including business intelligence scenarios in enterprises [31].

Consider the example Query 1 shown below.

Query 1

```
SELECT *
FROM R, S, T
WHERE R.a = S.b AND S.c = T.d AND T.e = 10
```

Figure 1.1 shows the major steps in the workflow of processing a SQL query in a RDBMS. The three stages of query processing are explained below.

Parsing and validation The parsing and validation step converts the input SQL query into an internal representation. This step ensures that

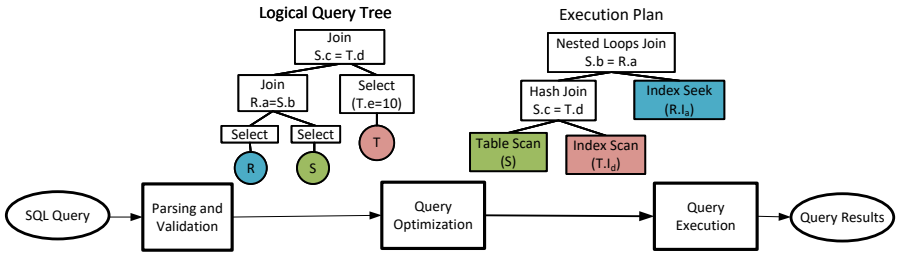


Figure 1.1: Workflow of query processing

the query adheres to the SQL syntax and only contains references to existing database objects, e.g., tables and columns. The output of this step is a logical query tree, an algebraic representation of the query in the form of a tree of logical relational operators (e.g,. Select, Join). For example, Figure 1.1 shows the output logical query tree of Query 1 after the parsing and validation step.

Query optimization The *query optimizer* takes a logical query tree as the input, and is responsible for generating an *efficient* execution plan that is either interpreted or compiled by the query execution engine. An *execution plan* (also referred to as *plan*) is a tree of physical operators, with edges representing the data flow between the operators. For example, Figure 1.1 shows the output execution plan of Query 1 after the query optimization step. For a given query, the number of different execution plans that may be used to answer the query may grow exponentially with the number of tables referenced in the query, and different execution plans can vary widely in terms of efficiency. Therefore, the performance of a query crucially depends on the ability of the optimizer to choose a good execution plan from a large space of alternatives. An overview of query optimization in RDBMSs is available in [28].

Query execution The query execution engine takes the plan from the query optimizer and executes the plan to produce the query results. The query execution engine implements a set of *physical operators*, which are building blocks for executing SQL query plans. A physical operator

takes one or more sets of data records as its input, referred to as *rows*, and outputs a set of rows. Examples of physical operators include Table Scan, Index Scan, Index Seek (see Appendix), Hash Join, Nested Loops Join, Merge Join, and Sort. For descriptions of algorithms used for various physical operators, we refer the reader to [77].

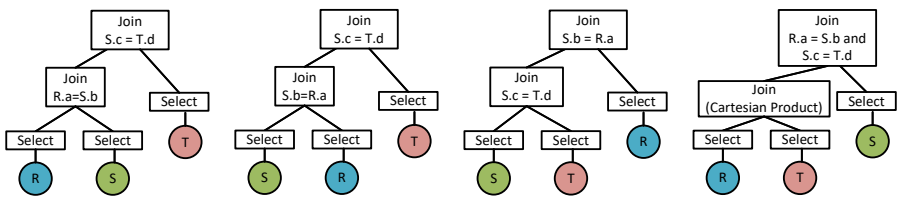
Query execution in a majority of relational database systems follows the *iterator* model, where each physical operator implements the *Open*, *GetNext*, and *Close* methods. Every iterator contains record of its state with information such as the size and the location of the hash table. In *Open*, the operator initializes its state and prepares for processing. When *GetNext* is called, the operator produces the next output row or indicates that there are no more rows, i.e., end of processing. We observe that to produce an output row a non-leaf operator in the plan needs to call *GetNext* on its child operator(s). For example, consider the execution plan shown in Figure 1.1. The Nested Loops Join operator calls *GetNext* on the Hash Join operator, which in turn calls *GetNext* on Table Scan(S) operator. When an operator completes producing its output rows (i.e., indicates that there are no more rows), the parent calls *Close* on it to allow the operator to clean up its state. The above approach of specifying operators through the iterator model makes it convenient to add new operators to the execution engine. Since each operator is an iterator from which rows are 'pulled', this model of execution is also referred to as a *pull model*. We refer the reader to [79] for a complete description of the pull model of query execution.

The iterator model as described above incurs high overhead of function invocations with each *GetNext* call processing a single row at a time, resulting in poor performance on modern CPUs. *Vectorization* enables batching so that a single *GetNext* call for a physical operator produces results for a batch of rows and leverages the SIMD instructions of modern CPUs [19]. Together with columnar representation [188], vectorization sharply increases the efficiency of query execution engines for decision support queries. In addition, *code generation* is a technique that generates efficient code from the query execution plan in a language such as C [152], which is then compiled and executed, or directly generates efficient machine code using a compiler framework such as LLVM [114]. The tradeoffs in vectorization and compilation are discussed in [107].

1.1 Key Challenges in Query Optimization

To choose an efficient plan among many alternative execution plans, a query optimizer must determine the *search space* of plans it will explore, compare the relative efficiency of the plans with *cost estimation*, and navigate the search space with an efficient *search algorithm* to find an execution plan that has very low (ideally lowest) cost of execution among its choices. We now briefly describe these facets of a query optimizer.

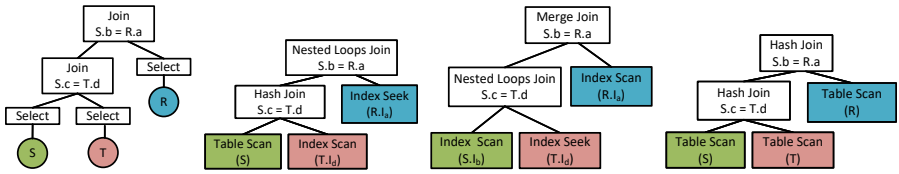
Search space The search space consists of alternative equivalent execution plans of the query, which can be large for complex queries. First, a given algebraic representation of a query can potentially be transformed into many other equivalent representations. These equivalences arise from properties of relational algebra, e.g., $Join(Join(R, S), T) \iff Join(Join(S, T), R)$ since the Join operator is commutative and associative [63]. Figure 1.2 shows four different but equivalent algebraic representations of the same query.



(a) Logical plan L_1 **(b)** Logical plan L_2 **(c)** Logical plan L_3 **(d)** Logical plan L_4

Figure 1.2: Semantically equivalent logical query trees

Second, for a given logical operator there are many different implementations of that logical operator. Hence, for a given logical query tree, there are potentially many different possible execution plans. For example, in Figure 1.3, for the logical query tree in Figure 1.3a, we show three out of many possible execution plans in Figure 1.3b-1.3d. Although the three plans have the same order in which joins are evaluated, they vary in the specific physical operators used to implement the logical operators. For example, the Select operator in Figure 1.3a can be implemented using Table Scan, Index Scan, or Index Seek; and the Join operator can be implemented using Nested Loops Join, Hash



(a) Logical plan L **(b)** Execution plan P_1 **(c)** Execution plan P_2 **(d)** Execution plan P_3

Figure 1.3: Different execution plans for a given logical query tree

Join, or Merge Join. The Nested Loops Join in Figure 1.3b may be the most efficient among the three when the *join size* (i.e., number of rows produced by the join) of the join between S and T is small and an index I_a is available on the join column $R.a$. The plan in Figure 1.3c with the Merge Join may be a good choice when an index I_b is available on $S.b$ and an index I_a is available on $R.a$, i.e., the indexes provide the sort order required by the Merge Join. In contrast, the plan in Figure 1.3d with the two Hash Join operators may be the plan of choice when the size of the join between S and T is large. Thus, unless the optimizer considers each of these plans in its search space and compares their resource usage and expected relative performance, it may not produce a good plan.

Cost estimation The efficiency of different execution plans for the same query, measured by their elapsed time or resources consumed (e.g., CPU, memory, I/O), can vary significantly, as the example in Figure 1.3 shows. The difference in elapsed time between a good and a poor execution plan for complex queries on large databases can be several orders of magnitudes. Therefore, to pick a good execution plan for a query from the space of execution plans as noted above, most query optimizers leverage a *cost model* that estimates the work done by query execution plans with sufficient fidelity so that relative comparisons of the execution plans are accurate. Specifically, a physical operator must estimate the work done by the algorithm used to implement that operator, and this estimation requires the sizes and other statistical characteristics of the input relation(s) to that operator as well as those of its output. Finally, even though the cost has at least three dimensions

(CPU, memory, I/O), the cost model combines these multi-dimensional costs in a single number for the convenience of comparing any two plans.

Search algorithm In principle, one could exhaustively enumerate every alternative execution plan in the search space and invoke cost estimation to determine the cost of each plan in order to find the plan with the lowest estimated cost. As some of the alternative execution plans can share common logical or physical operator trees, e.g., *Select(S)* in L_1-L_4 of Figure 1.2 or *Table Scan(S)* in P_1 and P_3 of Figure 1.3, the enumeration needs to be done carefully to avoid duplicate explorations. Even so, the exhaustive enumeration can still be too costly in practice. Thus, a good query optimizer will try to reduce the cost of enumeration without compromising significantly the quality of the chosen execution plan.

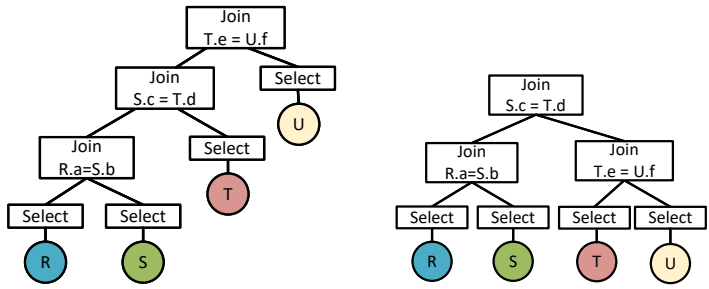
In summary, a good optimizer is one which: (a) considers a sufficiently large search space of promising plans, (b) models the cost of execution plans sufficiently accurately to distinguish between plans with significantly different costs, and (c) provides a search algorithm that efficiently finds a plan with low cost.

1.2 System R Query Optimizer

The System R project from IBM Research did pioneering work on query optimization [178]. We briefly review how the System R query optimizer addressed the key challenges mentioned in Section 1.1. The techniques developed in this project have had significant impact on all query optimizers that followed, including extensible query optimizers.

Search space The System R query optimizer's cost-based plan selection technique focused on the Select-Project-Join (SPJ) class of queries. The physical operators for implementing a Select operation included Table Scan and Index Scan. For Join, System R provided two physical operators, Nested Loops Join and Merge Join (which requires both inputs to be sorted on the respective join columns). In the example of Query 1, as Figure 1.2 and Figure 1.3 illustrate, there are several logical query

trees and execution plans for this SPJ queries. This arises because join is associative and commutative, and there are multiple options of physical operators for Scan and Join operations. The space of logical query trees explored by System R for SPJ queries included the space of *linear sequence* of binary Join operations, e.g., $Join(Join(Join(R, S), T), U)$. Figure 1.4a shows an example logical query tree of a linear sequence of Join operations whereas the logical query tree in Figure 1.4b, i.e., a bushy plan, is not in the search space of System R. The optimizer also offered techniques to improve the efficiency of nested queries based on program analysis but these optimizations were not cost-based.



(a) Linear plan **(b)** Bushy plan

Figure 1.4: Linear sequence of joins vs. bushy join

Cost model The cost model of System R used formulas to estimate the CPU and I/O costs for each operator in execution plans. Unlike today's optimizers, it did not incorporate the cost of memory. The System R optimizer maintained a set of statistics on base tables and indexes, e.g., number of rows (cardinality) and data pages in the table, number of pages in the index, number of distinct values in each column. System R provided a set of formulas to compute the selectivity of a single selection or join predicate. The selectivity of a WHERE clause containing a conjunction of selection predicates was computed by multiplying the selectivity of all predicates, i.e., assuming the predicates are independent. Thus, the cardinality of the output size of a join was estimated by taking the product of the cardinalities of the two input relations and multiplying it with the selectivity of all predicates. The cost model formulas, together with statistical information on base tables and indexes, enabled the

System R optimizer to perform estimation of CPU and I/O costs of execution plans.

Search algorithm The search algorithm of the System R optimizer used *dynamic programming* to find the "best" join order, and is based on the assumption that the cost model satisfies the *principle of optimality*. In other words, it assumes that, in the search space of linear sequence of joins, the optimal plan for a join of n relations can be found by extending the optimal plan of a sub-expression of $n-1$ joins with an additional join. For example, the optimal plan P_{RST} of joining relations R, S, and T can be found from joining R with P_{ST}, joining S with P_{RT}, and joining T with P_{RS}, where P_{ST}, P_{RT}, and P_{RS} are the optimal plans for joining S, T, joining R, T, and joining R, S respectively. In contrast to the naive approach that enumerates $O(n!)$ plans by enumerating all permutations of the join ordering, the dynamic programming approach enumerates $O(n2^{n-1})$ plans, and is therefore significantly faster, even though the time complexity is still exponential in the number of joins.

A second important aspect of System R's search algorithm was its consideration of *interesting orders*. Consider a query Q that joins three tables R, S, and T, with join predicates $R.a = S.a$ and $S.a = T.a$. Suppose the cost of joining R and S with Nested Loops Join using an Index Seek on S is smaller than the cost of using Merge Join. In this case, when considering plans for joining R, S, and T, the optimizer would prune out the plan where R and S are joined using Merge Join. However, if Merge Join is used to join R and S, then the result of the join is sorted on column a, which may significantly reduce the cost of the join with T if Merge Join is used. Therefore, pruning a plan that joins R and S with a Merge Join can result in a sub-optimal plan for the query. The fact that the output rows of an operator are ordered, i.e., the operator has an interesting order, may lower the cost of parent or ancestor operators in the plan. To accommodate this violation of the principle of optimality due to interesting orders while retaining the benefits of using dynamic programming, the search algorithm considered the interesting order for every expression it enumerates. For a join expression, plans were compared in cost if and only if they had the same interesting order, and an optimal plan was kept for each distinct interesting order.

1.3 Need for Extensible Query Optimizer Architecture

The important concepts introduced by System R, including the use of data statistics and a cost model to determine an execution plan, the dynamic programming based search for join ordering, and the need to consider interesting orders, have been adopted by virtually all widely used query optimizers. However, the framework could not be flexibly and efficiently extended to additional algebraic equivalences in relational algebra and new constructs in database systems in a cost-based manner, which can potentially miss out opportunities to find cheaper query plans. As relational databases and SQL became important for decision support queries, the transformations for these additional algebraic equivalences became valuable for generating an efficient execution plan. Examples of such transformations include pushing down a group-by below a join to reduce the cost of the join, optimization of outer joins that are not associative nor commutative, and decorrelation of nested queries. In addition, new constructs were introduced to database systems to improve query execution performance. For example, materialized views [43, 84], which precompute and store the results of a query subexpression, and thereby could dramatically reduce the cost of executing the query, became important for OLAP and other analytical workloads. Furthermore, the optimizer also needed to support new logical and physical operators that were introduced to efficiently execute SQL queries, e.g., Apply [69].

Fortunately, as the practical needs of a SQL query optimizer expanded, the research on extensible database systems that was ongoing at that time yielded architectural alternatives to extending the architecture of System R. Extensible database systems were envisioned as systems that can be used to customize a database system to the needs of an application. Specifically, Exodus [26] and later Volcano [79], which were designed to support user-specified operators for query execution, emerged in that context. Given the need to support custom operators, providing a framework for extensible query optimization became a necessity. Thus, extensibility of the optimizer was a design feature in Volcano from the very beginning as it was initially envisioned as an "experimental vehicle for multitude of purposes" [79]. They allowed

system designers to "plug-in" new *rules*, drawing inspiration from rules in expert systems (production systems), and thereby extend the capabilities of the optimizer. Later, the extensible optimizer frameworks of Volcano/Cascades [80, 82] and Starburst [123, 165] focused on SQL query optimization as a key application, which fulfilled a pressing need for a new architecture for SQL query optimization.

For most of this monograph, we will focus on extensible optimizers based on Volcano/Cascades. These extensible optimization frameworks center around the concept of rules. A *logical transformation rule* represents an equivalence in the SQL language (or its algebraic representation). For example, the equivalences implied by join commutativity and associativity noted earlier can be expressed using rules. Similarly, a rule may define the conditions under which pushing down a group-by operation below a join preserves equivalence. Applying logical transformation rules to a query tree results in an equivalent alternative query tree. An *implementation rule* defines the mapping from a logical operator (e.g., Join) to a physical operator (e.g., Hash Join). Implementation rules are needed to generate execution plans for the query. A judicious choice of a sequence of applications of rules can potentially transform the query tree into one that executes much faster. It should be noted that in this architecture, new operators, logical transformations, and implementation rules can be incorporated without having to modify the search algorithm of the optimizer each time. Last but not the least, it is important to note that transformations do not necessarily reduce cost, and therefore the search algorithm must choose among the alternatives in a cost-based manner.

SQL is a declarative query language. This allows the query optimizers to create efficient execution plans for SQL queries that leverage logical transformations that preserve semantic equivalence and also judiciously choose the most efficient implementation for the logical operators. The holy grail of query optimization is to produce the most efficient execution plan that preserves semantic equivalence but is independent of how the query is expressed syntactically by the users or the applications. The extensible query optimizers make this goal achievable by applying rules, chosen from a rich set of transformations, to the query tree successively in a judicious sequence driven by a cost-based search algorithm.

1.4 Outline

In this monograph, we focus on the technology of extensible query optimizers and use Microsoft SQL Server for illustration of the key concepts. In comparison to the overview article on query optimization by one of the authors [28], this monograph provides a detailed description of extensible optimizer frameworks as well as several additional transformation rules that are commonly used in practice. The extensibility framework and rules are explained in depth using pseduocode and examples.

The rest of the monograph is organized as follows:

Section 2: We review the extensible optimizer frameworks of Volcano [82] and its successor, the Cascades framework [80], that have been influential. We describe the search algorithms and key data structures needed in both frameworks, as well as additional techniques to improve the efficiency of query optimization. We illustrate how Microsoft SQL Server's query optimizer leverages the Cascades framework with a few examples. Finally, we describe how the optimizer handles parallel and distributed query processing.

Section 3: We present a brief review of other extensible query optimizers, including Starburst used in IBM DB2, Orca used in Greenplum DB, Calcite used in Apache Hive, and Catalyst used in Spark SQL. Although PostgreSQL's query optimizer does not possess the extensibility capabilities of frameworks such as Volcano and Cascades, given its popularity, we include a short overview of its query optimizer.

Section 4: An extensible optimizer draws its effectiveness from the rules it leverages. In this section, we review some of the key logical transformations and implementation rules relevant for access paths to base tables, inner and outer joins, group-by, aggregation, and decorrelation of nested queries. We touch upon a few selected "advanced" rules, e.g., for optimizing star and snowflake queries which are common in data warehouses, sideways information passing, user-defined functions (UDFs), and materialized views.

Section 5: An optimizer framework critically depends on the cost model and cardinality estimation. In this section, we provide an overview of cost modeling and cardinality estimation with a focus on industrial

practices. We discuss the statistical summaries used by the optimizer such as histograms and how they are used for complex queries. In addition, we discuss recent adoption of sampling and sketches in database systems. Finally, we illustrate these concepts and techniques using Microsoft SQL Server.

Section 6: Most articles on query optimization omit discussions on managing plans generated by the optimizer over the lifetime of the database. These aspects of plan management can critically impact overall workload performance. We discuss a few important challenges in this context: (a) plan caching and invalidation (b) improving sub-optimal plans with execution feedback (c) query hints, which allow users to influence the plan that is chosen by the optimizer (d) optimizing parameterized queries.

Section 7: While this monograph is centered on extensible query optimizers in practice, we use this section to mention some of the open problems and a few of research directions that are being pursued.

Errata and updates: We will provide corrections and updates to this monograph at the following URL [59]. We encourage readers who discover errors in this monograph to report them to the authors via email.

1.5 Suggested Reading

Citation numbers below correspond to numbers in the References section.

[178] P. G. Selinger *et al.*, "Access Path Selection in a Relational Database Management System," in *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '79, pp. 23–34, Boston, Massachusetts: Association for Computing Machinery, 1979. DOI: 10.1145/582095.582099

[77] G. Graefe, "Query Evaluation Techniques for Large Databases," *ACM Computing Surveys (CSUR)*, vol. 25, no. 2, 1993, pp. 73–169

[28] S. Chaudhuri, "An Overview of Query Optimization in Relational Systems," in *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pp. 34–43, 1998

Appendix

A

Access Methods

We provide a brief overview of how the query execution engine in relational databases can access data stored in the base tables. The data in base tables can be physically organized using different persistent (i.e., on-disk) data structures. Some of the most commonly used structures are heaps, B-trees indexes [14], and columnstore indexes [189]. We use the examples of heaps and B+-tree indexes to introduce the important physical operators.[1] We note that there are other aspects of access methods on base tables that are not discussed below, e.g., partitioning, but are also relevant for query optimization.

Heap and B+-tree index A heap is an unordered collection of all records in the table. Each record (row) has a DBMS generated row id, and stores values for each column in the table. Rows are organized into pages and stored on disk. B+-trees are n-ary tree based data structures that organize the data ordered by the *key columns* of the index. Further, a B+-tree index can either be a clustered index or non-clustered index. In a clustered index, the leaf pages of the index contain the entire record

[1] In contrast to a B-tree, in a B+-tree, leaf pages contain a pointer to the next leaf page in index order, thereby enabling more efficient scans of a range of values.

(i.e., values of all columns of the table), whereas in a non-clustered index, the leaf pages only contain the key columns and the record id. Besides key columns, a non-clustered index may optionally contain additional *include* columns. In this case, each row in the leaf page of the index contains the key columns as well as the include columns. Observe that the B+-tree supports search (i.e., lookups or range scans) only over the key columns, and not the include columns. In contrast to heaps, B+-Tree indexes can greatly speed up the retrieval of the data, especially when only a selective subset of the data is needed to answer the query.

We use the same example table and query from Section 4.1 to describe the physical operators for accessing data in heaps and B+-tree indexes. Consider a table $S(id, a, b, c)$ with four columns, where id is the *primary key* of table S. Consider the following query Q_1:

```
SELECT S.a, S.b
FROM S
WHERE S.a > 10 AND S.b = 20
```

Table Scan Since the rows of a heap are unordered, a heap provides no ability to lookup any individual record. Thus, the only physical operator allowed on a heap is the Table Scan operator. Table Scan takes a table as an argument and returns all rows from the table. In our example, Table Scan of S returns all rows in S. Observe that when the table is large, the Table Scan operator can be expensive since it needs to fetch all pages of S from storage, including rows and columns that are not needed to answer the query.

Index Seek and Key Lookup When the query contains an equality predicate on any prefix of the key columns in the index, the Index Seek operator can be used to retrieve all rows satisfying the predicate. For example, consider a B+-tree index I_b built on table S with b as the key column. Then for Q_1, instead of scanning the full table, invoking Index Seek (I_b, $S.b = 20$) will find and retrieve row ids of all rows in the table satisfying the predicate. A special case of Index Seek is a Key Lookup operator which is used when the index is defined on a primary key or unique column of the table. In a Key Lookup, either 0 or 1 record is returned, whereas in an Index Seek, 0 or more rows

can be returned. Note that an invocation of the Index Seek operator performs a random I/O to access the data page containing the matching records. While an Index Seek is often used for identifying rows satisfying a selection predicates (e.g., $S.b = 10$), it can also be combined with a Nested Loops Join operator to efficiently support a join between two relations. Specifically, for each row from the outer relation, a Nested Loops Join operator can use an Index Seek on the inner side relation of the join if the key column of the index is the join column.

Index Scan When the query contains a range predicate, a B+-tree index enables efficient range scans using the Index Scan operator. Consider a B+-tree index I_a built on table S with a as the key column. Since the predicate $S.a > 10$ needs to retrieve a range of values, invoking Index Scan (I_a, $S.a > 10$) will retrieve row ids of all rows satisfying the range predicate on column a. We observe that for the Index Scan operators the predicate is an optional argument. If no predicate is specified, Index Scan returns all rows from the leaf pages of the index. Unlike Table Scan where the rows returned are unordered, these rows from Index Scan will appear in the order of the key columns of the index. Thus, the usefulness of Index Scan goes beyond its ability to retrieve rows since it can benefit other operators such as Merge Join and Stream Aggregate, which require their inputs to be sorted. B+-tree indexes containing include columns can be very effective in answering a query when all columns required to answer the query are available in the index, whether as part of key or include columns. For example, consider an index $I_a(b)$ where the key column is a, and the include column is b. Observe that the query Q_1 can be answered using an Index Scan $I_a(b)$ with $S.a > 10$ followed by a Filter operator that can apply the predicate $S.b = 20$. Since the column b is available in the index, we can avoid a Key Lookup into the clustered index to obtain column b.

References

[1] D. J. Abadi, S. R. Madden, and N. Hachem, "Column-stores vs. Row-stores: How Different are They Really?" In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '08, pp. 967–980, Vancouver, Canada: Association for Computing Machinery, 2008. DOI: 10.1145/1376616.1376712.

[2] A. Aboulnaga and S. Chaudhuri, "Self-Tuning Histograms: Building Histograms without Looking at Data," in *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '99, pp. 181–192, Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1999. DOI: 10.1145/304182.304198.

[3] P. K. Agarwal, G. Cormode, Z. Huang, J. M. Phillips, Z. Wei, and K. Yi, "Mergeable Summaries," *ACM Transactions on Database Systems (TODS)*, vol. 38, no. 4, 2013, pp. 1–28.

[4] J. Aguilar-Saborit, R. Ramakrishnan, K. Srinivasan, K. Bocksrocker, I. Alagiannis, M. Sankara, M. Shafiei, J. Blakeley, G. Dasarathy, S. Dash, *et al.*, "POLARIS: the Distributed SQL Engine in Azure Synapse," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, 2020, pp. 3204–3216.

[5] M. X. et al., "Adaptive and Robust Query Execution for Lakehouses at Scale," *Proceedings of the VLDB Endowment*, vol. 17, 2024.

[6] Amazon, *AWS: COUNT function*, 2024. URL: https://docs.aws.amazon.com/redshift/latest/dg/r_COUNT.html.

[7] G. Antoshenkov, "Dynamic Query Optimization in Rdb/VMS," in *Proceedings of IEEE 9th International Conference on Data Engineering*, IEEE, pp. 538–547, 1993.

[8] P. M. Aoki, "Implementation of Extended Indexes in POSTGRES," in *ACM SIGIR Forum*, ACM New York, NY, USA, vol. 25, pp. 2–9, 1991.

[9] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, *et al.*, "Spark SQL: Relational Data Processing in Spark," in *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pp. 1383–1394, 2015.

[10] N. Armenatzoglou, S. Basu, N. Bhanoori, M. Cai, N. Chainani, K. Chinta, V. Govindaraju, T. J. Green, M. Gupta, S. Hillig, *et al.*, "Amazon Redshift Re-invented," in *Proceedings of the 2022 International Conference on Management of Data*, pp. 2205–2217, 2022.

[11] A. Atserias, M. Grohe, and D. Marx, "Size Bounds and Query Plans for Relational Joins," in *Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pp. 739–748, 2008.

[12] R. Avnur and J. M. Hellerstein, "Eddies: Continuously Adaptive Query Processing," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pp. 261–272, 2000.

[13] B. Babcock and S. Chaudhuri, "Towards a Robust Query Optimizer: A Principled and Practical Approach," in *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '05, pp. 119–130, Baltimore, Maryland: Association for Computing Machinery, 2005. DOI: 10.1145/1066157.1066172.

[14] R. Bayer and E. McCreight, "Organization and Maintenance of Large Ordered Indices," in *Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, pp. 107–141, 1970.

[15] E. Begoli, J. Camacho-Rodríguez, J. Hyde, M. J. Mior, and D. Lemire, "Apache Calcite: A Foundational Framework for Optimized Query Processing over Heterogeneous Data Sources," in *Proceedings of the 2018 International Conference on Management of Data*, pp. 221–230, 2018.

[16] S. Bellamkonda, R. Ahmed, A. Witkowski, A. Amor, M. Zait, and C.-C. Lin, "Enhanced Subquery Optimizations in Oracle," *Proc. VLDB Endow.*, vol. 2, no. 2, Aug. 2009, pp. 1366–1377. DOI: 10.14778/1687553.1687563.

[17] P. A. Bernstein and D.-M. W. Chiu, "Using Semi-Joins to Solve Relational Queries," *J. ACM*, vol. 28, no. 1, Jan. 1981, pp. 25–40. DOI: 10.1145/322234.322238.

[18] P. Bizarro, N. Bruno, and D. J. DeWitt, "Progressive Parametric Query Optimization," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 4, 2009, pp. 582–594. DOI: 10.1109/TKDE.2008.160.

[19] P. A. Boncz, M. Zukowski, and N. Nes, "MonetDB/X100: Hyper-Pipelining Query Execution.," in *Cidr*, vol. 5, pp. 225–237, 2005.

[20] R. Borovica-Gajic, S. Idreos, A. Ailamaki, M. Zukowski, and C. Fraser, "Smooth Scan: Statistics-oblivious Access Paths," in *2015 IEEE 31st International Conference on Data Engineering*, IEEE, pp. 315–326, 2015.

[21] N. Bruno and S. Chaudhuri, "Exploiting Statistics on Query Expressions for Optimization," in *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pp. 263–274, 2002.

[22] N. Bruno, S. Chaudhuri, and L. Gravano, "STHoles: A Multidimensional Workload-Aware Histogram," in *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pp. 211–222, 2001.

[23] N. Bruno, S. Chaudhuri, and R. Ramamurthy, "Power Hints for Query Optimization," in *2009 IEEE 25th International Conference on Data Engineering*, pp. 469–480, 2009. DOI: 10.1109/ICDE.2009.68.

[24] N. Bruno, C. Galindo-Legaria, M. Joshi, E. Calvo Vargas, K. Mahapatra, S. Ravindran, G. Chen, E. Cervantes Juárez, and B. Sezgin, "Unified Query Optimization in the Fabric Data Warehouse," in *Companion of the 2024 International Conference on Management of Data*, pp. 18–30, 2024.

[25] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache Flink: Stream and Batch Processing in a Single Engine," *The Bulletin of the Technical Committee on Data Engineering*, vol. 38, no. 4, 2015.

[26] M. J. Carey, D. J. DeWitt, G. Graefe, D. M. Haight, J. E. Richardson, D. T. Schuh, E. J. Shekita, and S. L. Vandenberg, "The EXODUS Extensible DBMS project: An Overview," 1988.

[27] M. Charikar, S. Chaudhuri, R. Motwani, and V. Narasayya, "Towards Estimation Error Guarantees for Distinct Values," in *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 268–279, 2000.

[28] S. Chaudhuri, "An Overview of Query Optimization in Relational Systems," in *Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pp. 34–43, 1998.

[29] S. Chaudhuri, "Query Optimizers: Time to Rethink the Contract?" In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pp. 961–968, 2009.

[30] S. Chaudhuri, G. Das, and U. Srivastava, "Effective use of Block-level Sampling in Statistics Estimation," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pp. 287–298, 2004.

[31] S. Chaudhuri, U. Dayal, and V. Narasayya, "An Overview of Business Intelligence Technology," *Communications of the ACM*, vol. 54, no. 8, 2011, pp. 88–98.

[32] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim, "Optimizing Queries with Materialized Views," in *Proceedings of the Eleventh International Conference on Data Engineering*, IEEE, pp. 190–200, 1995.

[33] S. Chaudhuri, H. Lee, and V. R. Narasayya, "Variance aware Optimization of Parameterized Queries," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pp. 531–542, 2010.

[34] S. Chaudhuri, R. Motwani, and V. Narasayya, "Random Sampling for Histogram Construction: How Much is Enough?" *ACM SIGMOD Record*, vol. 27, no. 2, 1998, pp. 436–447.

[35] S. Chaudhuri, R. Motwani, and V. Narasayya, "On Random Sampling over Joins," *ACM SIGMOD Record*, vol. 28, no. 2, 1999, pp. 263–274.

[36] S. Chaudhuri and V. Narasayya, "Automating Statistics Management for Query Optimizers," *IEEE Transactions on Knowledge and Data Engineering*, vol. 13, no. 1, 2001, pp. 7–20.

[37] S. Chaudhuri, V. Narasayya, and R. Ramamurthy, "A Pay-as-You-Go Framework for Query Execution Feedback," *Proc. VLDB Endow.*, vol. 1, no. 1, Aug. 2008, pp. 1141–1152. DOI: 10.14778/1453856.1453977.

[38] S. Chaudhuri and K. Shim, "Including Group-By in Query Optimization," in *Proceedings of the 20th International Conference on Very Large Data Bases*, ser. VLDB '94, pp. 354–366, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994.

[39] S. Chaudhuri and K. Shim, "Optimization of Queries with User-defined Predicates," *ACM Transactions on Database Systems (TODS)*, vol. 24, no. 2, 1999, pp. 177–228.

[40] C. M. Chen and N. Roussopoulos, "Adaptive Selectivity Estimation using Query Feedback," in *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pp. 161–172, 1994.

[41] T. Chen and C. Guestrin, "Xgboost: A Scalable Tree Boosting System," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

[42] A. Cheung, A. Solar-Lezama, and S. Madden, "Optimizing database-backed applications with query synthesis," *ACM SIGPLAN Notices*, vol. 48, no. 6, 2013, pp. 3–14.

[43] R. Chirkova, J. Yang, *et al.*, "Materialized Views," *Foundations and Trends® in Databases*, vol. 4, no. 4, 2011, pp. 295–405.

[44] F. Chu, J. Y. Halpern, and P. Seshadri, "Least Expected Cost Query Optimization: An Exercise in Utility," in *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pp. 138–147, 1999.

[45] R. L. Cole and G. Graefe, "Optimization of Dynamic Query Evaluation Plans," in *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '94, pp. 150–160, Minneapolis, Minnesota, USA: Association for Computing Machinery, 1994. DOI: 10.1145/191839.191872.

[46] G. Cormode, M. Garofalakis, P. J. Haas, C. Jermaine, *et al.*, "Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches," *Foundations and Trends® in Databases*, vol. 4, no. 1–3, 2011, pp. 1–294.

[47] G. Cormode and S. Muthukrishnan, "An Improved Data Stream Summary: the Count-Min Sketch and its Applications," *Journal of Algorithms*, vol. 55, no. 1, 2005, pp. 58–75.

[48] G. Cormode and S. Muthukrishnan, "An Improved Data Stream Summary: the Count-Min Sketch and its Applications," *Journal of Algorithms*, vol. 55, no. 1, 2005, pp. 58–75.

[49] H. D., P. N. Darera, and J. R. Haritsa, "On the Production of Anorexic Plan Diagrams," in *Proceedings of the 33rd International Conference on Very Large Data Bases*, ser. VLDB '07, pp. 1081–1092, Vienna, Austria: VLDB Endowment, 2007.

[50] B. Dageville, T. Cruanes, M. Zukowski, V. Antonov, A. Avanes, J. Bock, J. Claybaugh, D. Engovatov, M. Hentschel, J. Huang, *et al.*, "The Snowflake Elastic Data Warehouse," in *Proceedings of the 2016 International Conference on Management of Data*, pp. 215–226, 2016.

[51] S. Dar, M. J. Franklin, B. T. Jonsson, D. Srivastava, M. Tan, *et al.*, "Semantic Data Caching and Replacement," in *VLDB*, vol. 96, pp. 330–341, 1996.

[52] U. Dayal, "Of Nests and Trees: A Unified Approach to Processing Queries That Contain Nested Subqueries, Aggregates, and Quantifiers," in *Proceedings of the 13th International Conference on Very Large Data Bases*, ser. VLDB '87, pp. 197–208, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1987.

[53] K. Delaney, *Inside Microsoft SQL Server 2005: Query Tuning and Optimization*. Microsoft Press, 2007.

[54] A. Deshpande, Z. Ives, and V. Raman, "Adaptive Query Processing," *Foundations and Trends® in Databases*, vol. 1, no. 1, 2007, pp. 1–140. DOI: 10.1561/1900000001.

[55] D. J. DeWitt and R. Ramamurthy, "Buffer Pool Aware Query Optimization," in *Proceedings of the 2005 CIDR Conference*, pp. 961–968, 2009.

[56] D. J. DeWitt, J. F. Naughton, and D. A. Schneider, "An Evaluation of Non-Equijoin Algorithms," in *Proceedings of the 17th International Conference on Very Large Data Bases*, ser. VLDB '91, pp. 443–452, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1991.

[57] B. Ding, S. Chaudhuri, J. Gehrke, and V. Narasayya, "DSB: A Decision Support Benchmark for Workload-Driven and Traditional Database Systems," *Proc. VLDB Endow.*, vol. 14, no. 13, Sep. 2021, pp. 3376–3388. DOI: 10.14778/3484224.3484234.

[58] B. Ding, S. Chaudhuri, and V. Narasayya, "Bitvector-Aware Query Optimization for Decision Support Queries," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '20, pp. 2011–2026, Portland, OR, USA: Association for Computing Machinery, 2020. DOI: 10.1145/3318464.3389769.

[59] B. Ding, V. Narasayya, and C. Surajit, *Errata and Updates to the Foundations and Trends in Databases article: Extensible Query Optimizers in Practice*, 2024. URL: https://www.microsoft.com/en-us/research/project/extensible-query-optimizers-in-practice-errata-and-updates/.

[60] W. Du, R. Krishnamurthy, and M.-C. Shan, "Query Optimization in a Heterogeneous DBMS," in *VLDB*, vol. 92, pp. 277–291, 1992.

[61] A. Dutt, V. Narasayya, and S. Chaudhuri, "Leveraging Re-Costing for Online Optimization of Parameterized Queries with Guarantees," in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD '17, pp. 1539–1554, Chicago, Illinois, USA: Association for Computing Machinery, 2017. DOI: 10.1145/3035918.3064040.

[62] M. Elhemali, C. A. Galindo-Legaria, T. Grabs, and M. M. Joshi, "Execution Strategies for SQL Subqueries," in *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '07, pp. 993–1004, Beijing, China: Association for Computing Machinery, 2007. DOI: 10.1145/1247480.1247598.

[63] R. Fagin, "Normal Forms and Relational Database Operators," in *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '79, pp. 153–160, Boston, Massachusetts: Association for Computing Machinery, 1979. DOI: 10.1145/582095.582120.

[64] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier, "Hyperloglog: the Analysis of a Near-Optimal Cardinality Estimation Algorithm," in *Discrete Mathematics and Theoretical Computer Science*, Discrete Mathematics and Theoretical Computer Science, pp. 137–156, 2007.

[65] P. Flajolet and G. N. Martin, "Probabilistic Counting Algorithms for Data Base Applications," *Journal of computer and system sciences*, vol. 31, no. 2, 1985, pp. 182–209.

[66] M. Freitag, M. Bandle, T. Schmidt, A. Kemper, and T. Neumann, "Adopting Worst-case Optimal Joins in Relational Database Systems," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, 2020, pp. 1891–1904.

[67] M. Freitag and T. Neumann, "Every Row Counts: Combining Sketches and Sampling for Accurate Group-by Result Estimates," *ratio*, vol. 1, 2019, pp. 1–39.

[68] C. Galindo-Legaria and A. Rosenthal, "How to Extend a Conventional Optimizer to Handle One- and Two-sided Outerjoin," in *[1992] Eighth International Conference on Data Engineering*, pp. 402–409, 1992. DOI: 10.1109/ICDE.1992.213169.

[69] C. Galindo-Legaria and M. Joshi, "Orthogonal Optimization of Subqueries and Aggregation," SIGMOD '01, 2001, pp. 571–581. DOI: 10.1145/375663.375748.

[70] C. Galindo-Legaria and A. Rosenthal, "Outerjoin Simplification and Reordering for Query Optimization," *ACM Trans. Database Syst.*, vol. 22, no. 1, Mar. 1997, pp. 43–74. DOI: 10.1145/244810.244812.

[71] C. A. Galindo-Legaria, M. M. Joshi, F. Waas, and M.-C. Wu, "Statistics on Views," in *Proceedings 2003 VLDB Conference*, Elsevier, pp. 952–962, 2003.

[72] C. A. Galindo-Legaria, T. Grabs, S. Gukal, S. Herbert, A. Surna, S. Wang, W. Yu, P. Zabback, and S. Zhang, "Optimizing Star Join Queries for Data Warehousing in Microsoft SQL Server," in *2008 IEEE 24th International Conference on Data Engineering*, pp. 1190–1199, 2008. DOI: 10.1109/ICDE.2008.4497528.

[73] S. Ganguly, "Design and Analysis of Parametric Query Optimization Algorithms," in *VLDB*, vol. 98, pp. 228–238, 1998.

[74] P. B. Gibbons, Y. Matias, and V. Poosala, "Fast Incremental Maintenance of Approximate Histograms," in *VLDB*, vol. 97, pp. 466–475, 1997.

[75] P. B. Gibbons, Y. Matias, and V. Poosala, "Fast Incremental Maintenance of Approximate Histograms," *ACM Transactions on Database Systems (TODS)*, vol. 27, no. 3, 2002, pp. 261–298.

[76] J. Goldstein and P.-Å. Larson, "Optimizing Queries using Materialized Views: a Practical, Scalable Solution," *ACM SIGMOD Record*, vol. 30, no. 2, 2001, pp. 331–342.

[77] G. Graefe, "Query Evaluation Techniques for Large Databases," *ACM Computing Surveys (CSUR)*, vol. 25, no. 2, 1993, pp. 73–169.

[78] G. Graefe, "Query Evaluation Techniques for Large Databases," *ACM Comput. Surv.*, vol. 25, no. 2, Jun. 1993, pp. 73–169. DOI: 10.1145/152610.152611.

[79] G. Graefe, "Volcano - An Extensible and Parallel Query Evaluation System," *IEEE Transactions on Knowledge and Data Engineering*, vol. 6, no. 1, 1994, pp. 120–135.

[80] G. Graefe, "The Cascades Framework for Query Optimization," *IEEE Data Eng. Bull.*, vol. 18, no. 3, 1995, pp. 19–29.

[81] G. Graefe, "New Algorithms for Join and Grouping Operations," *Computer Science-Research and Development*, vol. 27, 2012, pp. 3–27.

[82] G. Graefe and W. McKenna, "The Volcano Optimizer Generator," Colorado Univ at Boulder Dept of Computer Science, Tech. Rep., 1991.

[83] G. Graefe and W. J. McKenna, "The Volcano Optimizer Generator: Extensibility and Efficient Search," in *Proceedings of IEEE 9th international conference on data engineering*, pp. 209–218, 1993.

[84] A. Gupta and I. S. Mumick, *Materialized Views: Techniques, Implementations, and Applications*. MIT press, 1999.

[85] A. Gupta, I. S. Mumick, *et al.*, "Maintenance of Materialized Views: Problems, Techniques, and Applications," *IEEE Data Eng. Bull.*, vol. 18, no. 2, 1995, pp. 3–18.

[86] L. M. Haas, J. C. Freytag, G. M. Lohman, and H. Pirahesh, "Extensible Query Processing in Starburst," in *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '89, pp. 377–388, Portland, Oregon, USA: Association for Computing Machinery, 1989. DOI: 10.1145/67544.66962.

[87] A. Y. Halevy, "Answering Queries using Views: A Survey," *The VLDB Journal*, vol. 10, 2001, pp. 270–294.

[88] J. Haritsa, "Robust query processing: A survey," *Foundations and Trends® in Databases*, vol. 15, no. 1, 2024, pp. 1–114. URL: https://nowpublishers.com/article/Details/DBS-089.

[89] J. R. Haritsa, "Robust Query Processing: Mission Possible," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, IEEE, pp. 2072–2075, 2019.

[90] H. Harmouch and F. Naumann, "Cardinality Estimation: An Experimental Survey," *Proceedings of the VLDB Endowment*, vol. 11, no. 4, 2017, pp. 499–512.

[91] J. M. Hellerstein and M. Stonebraker, "Predicate Migration: Optimizing Queries with Expensive Predicates," in *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pp. 267–276, 1993.

[92] S. Heule, M. Nunkesser, and A. Hall, "Hyperloglog in Practice: Algorithmic Engineering of a state of the art Cardinality Estimation Algorithm," in *Proceedings of the 16th International Conference on Extending Database Technology*, pp. 683–692, 2013.

[93] E. Hewitt, *Cassandra: the Definitive Guide*. O'Reilly Media, Inc., 2010.

[94] B. Hilprecht and C. Binnig, "Zero-shot Cost Models for Out-of-the-box Learned Cost Prediction," *arXiv preprint arXiv:2201.00561*, 2022.

[95] B. Hilprecht, A. Schmidt, M. Kulessa, A. Molina, K. Kersting, and C. Binnig, "Deepdb: Learn from Data, not from Queries!" *arXiv preprint arXiv:1909.00607*, 2019.

[96] M. Hirzel, R. Soulé, S. Schneider, B. Gedik, and R. Grimm, "A Catalog of Stream Processing Optimizations," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, 2014, pp. 1–34.

[97] Y. Huai, A. Chauhan, A. Gates, G. Hagleitner, E. N. Hanson, O. O'Malley, J. Pandey, Y. Yuan, R. Lee, and X. Zhang, "Major Technical Advancements in Apache Hive," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pp. 1235–1246, 2014.

[98] A. Hulgeri and S. Sudarshan, "Parametric Query Optimization for Linear and Piecewise Linear Cost Functions," in *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, Elsevier, pp. 167–178, 2002.

[99] IBM, *DB2 for z/OS: Histogram statistics*, 2024. URL: https://www.ibm.com/docs/en/db2-for-zos/12?topic=statistics-histogram.

[100] Y. E. Ioannidis and Y. Kang, "Randomized Algorithms for Optimizing Large Join Queries," in *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '90, pp. 312–321, Atlantic City, New Jersey, USA: Association for Computing Machinery, 1990. DOI: 10.1145/93597.98740.

[101] Y. Ioannidis, "The History of Histograms (abridged)," in *Proceedings 2003 VLDB Conference*, Elsevier, pp. 19–30, 2003.

[102] Y. E. Ioannidis and S. Christodoulakis, "On the Propagation of Errors in the Size of Join Results," in *Proceedings of the 1991 ACM SIGMOD International Conference on Management of data*, pp. 268–277, 1991.

[103] Y. E. Ioannidis and V. Poosala, "Balancing Histogram Optimality and Practicality for Query Result Size Estimation," *Acm Sigmod Record*, vol. 24, no. 2, 1995, pp. 233–244.

[104] Y. Izenov, A. Datta, F. Rusu, and J. H. Shin, "COMPASS: Online Sketch-based Query Optimization for In-memory Databases," in *Proceedings of the 2021 International Conference on Management of Data*, pp. 804–816, 2021.

[105] N. Kabra and D. J. DeWitt, "Efficient Mid-query Re-optimization of Sub-optimal Query Execution Plans," in *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pp. 106–117, 1998.

[106] S. Kandula, L. Orr, and S. Chaudhuri, "Pushing data-induced predicates through joins in big-data clusters," *Proceedings of the VLDB Endowment*, vol. 13, no. 3, 2019, pp. 252–265.

[107] T. Kersten, V. Leis, A. Kemper, T. Neumann, A. Pavlo, and P. Boncz, "Everything you always wanted to know about Compiled and Vectorized Queries but were afraid to ask," *Proceedings of the VLDB Endowment*, vol. 11, no. 13, 2018, pp. 2209–2222.

[108] K. Kim, J. Jung, I. Seo, W.-S. Han, K. Choi, and J. Chong, "Learned Cardinality Estimation: An In-Depth Study," in *Proceedings of the 2022 International Conference on Management of Data*, ser. SIGMOD '22, pp. 1214–1227, Philadelphia, PA, USA: Association for Computing Machinery, 2022. DOI: 10.1145/3514221.3526154.

[109] W. Kim, "On Optimizing an SQL-like Nested Query," *ACM Trans. Database Syst.*, vol. 7, no. 3, Sep. 1982, pp. 443–469. DOI: 10.1145/319732.319745.

[110] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo Planning," in *European conference on machine learning*, Springer, pp. 282–293, 2006.

[111] R. P. Kooi, *The Optimization of Queries in Relational Databases*. Case Western Reserve University, 1980.

[112] P. Krishnan, J. S. Vitter, and B. Iyer, "Estimating Alphanumeric Selectivity in the Presence of Wildcards," in *Proceedings of the 1996 ACM SIGMOD international conference on Management of data*, pp. 282–293, 1996.

[113] P.-Å. Larson, C. Clinciu, E. N. Hanson, A. Oks, S. L. Price, S. Rangarajan, A. Surna, and Q. Zhou, "SQL Server Column Store Indexes," in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '11, pp. 1177–1184, Athens, Greece: Association for Computing Machinery, 2011. DOI: 10.1145/1989323.1989448.

[114] C. Lattner and V. Adve, "LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation," in *International symposium on code generation and optimization, 2004. CGO 2004.*, IEEE, pp. 75–86, 2004.

[115] A. W. Lee and M. Zait, "Closing the Query Processing Loop in Oracle 11g," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, 2008, pp. 1368–1378.

[116] K. Lee, A. Dutt, V. Narasayya, and S. Chaudhuri, "Analyzing the Impact of Cardinality Estimation on Execution Plans in Microsoft SQL Server," *Proceedings of the VLDB Endowment*, vol. 16, no. 11, 2023, pp. 2871–2883.

[117] M. K. Lee, J. C. Freytag, and G. M. Lohman, "Implementing an Interpreter for Functional Rules in a Query Optimizer.," in *VLDB*, pp. 218–229, 1988.

[118] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann, "How Good Are Query Optimizers, Really?" *Proc. VLDB Endow.*, vol. 9, no. 3, Nov. 2015, pp. 204–215. DOI: 10.14778/2850583.2850594.

[119] V. Leis, B. Radke, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann, "Query Optimization Through the Looking Glass, and What We Found Running the Join Order Benchmark," *The VLDB Journal*, vol. 27, 2018, pp. 643–668.

[120] W. Lemahieu, S. vanden Broucke, and B. Baesens, *Principles of Database Management: the Practical Guide to Storing, Managing and Analyzing Big and Small Data*. Cambridge University Press, 2018.

[121] B. Li, Y. Lu, and S. Kandula, "Warper: Efficiently Adapting Learned Cardinality Estimators to Data and Workload Drifts," in *Proceedings of the 2022 International Conference on Management of Data*, pp. 1920–1933, 2022.

[122] G. Lohman, *Is Query Optimization a "Solved" Problem?* 2014. URL: https://wp.sigmod.org/?p=1075.

[123] G. M. Lohman, "Grammar-like Functional Rules for Representing Query Optimization Alternatives," *ACM SIGMOD Record*, vol. 17, no. 3, 1988, pp. 18–27.

[124] Y. Lu, S. Kandula, A. C. König, and S. Chaudhuri, "Pre-training Summarization Models of Structured Datasets for Cardinality Estimation," *Proceedings of the VLDB Endowment*, vol. 15, no. 3, 2021, pp. 414–426.

[125] L. F. Mackert and G. M. Lohman, "R* Optimizer Validation and Performance Evaluation for Local Queries," in *Proceedings of the 1986 ACM SIGMOD international conference on Management of data*, pp. 84–95, 1986.

[126] L. F. Mackert and G. M. Lohman, "R* optimizer validation and performance evaluation for local queries," in *Proceedings of the 1986 ACM SIGMOD international conference on Management of data*, pp. 84–95, 1986.

[127] R. Marcus, P. Negi, H. Mao, N. Tatbul, M. Alizadeh, and T. Kraska, "Bao: Making Learned Query Optimization Practical," in *Proceedings of the 2021 International Conference on Management of Data*, ser. SIGMOD '21, pp. 1275–1288, Virtual Event, China: Association for Computing Machinery, 2021. DOI: 10.1145/3448016.3452838.

[128] R. Marcus, P. Negi, H. Mao, C. Zhang, M. Alizadeh, T. Kraska, O. Papaemmanouil, and N. Tatbul, "Neo: A Learned Query Optimizer," *arXiv preprint arXiv:1904.03711*, 2019.

[129] R. Marcus and O. Papaemmanouil, "Plan-structured Deep Neural Network Models for Query Performance Prediction," *arXiv preprint arXiv:1902.00132*, 2019.

[130] V. Markl, V. Raman, D. Simmen, G. Lohman, H. Pirahesh, and M. Cilimdzic, "Robust Query Processing through Progressive Optimization," in *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pp. 659–670, 2004.

[131] Y. Matias, J. S. Vitter, and M. Wang, "Wavelet-based Histograms for Selectivity Estimation," in *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pp. 448–459, 1998.

[132] W. J. McKenna, *Efficient Search in Extensible Database Query Optimization: The Volcano Optimizer Generator*. University of Colorado at Boulder, 1993.

[133] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, T. Vassilakis, H. Ahmadi, D. Delorey, S. Min, *et al.*, "Dremel: A Decade of Interactive SQL Analysis at Web Scale," *Proceedings of the VLDB Endowment*, vol. 13, no. 12, 2020, pp. 3461–3472.

[134] J. Melton and A. R. Simon, *SQL: 1999: Understanding Relational Language Components*. Elsevier, 2001.

[135] Microsoft, *Adaptive Joins in Microsoft SQL Server*, 2017. URL: https://learn.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing-details?view=sql-server-ver16.

[136] Microsoft, *Intro to Query Execution Bitmap Filters*, 2019. URL: https://techcommunity.microsoft.com/t5/sql-server-blog/intro-to-query-execution-bitmap-filters/ba-p/383175.

[137] Microsoft, *Optimizing Your Query Plans with the SQL Server 2014 Cardinality Estimator*, 2021. URL: https://learn.microsoft.com/en-us/previous-versions/dn673537(v=msdn.10)?redirectedfrom=MSDN.

[138] Microsoft, *Parameter Sensitive Plan optimization in Microsoft SQL Server*, 2022. URL: https://learn.microsoft.com/en-us/sql/relational-databases/performance/parameter-sensitive-plan-optimization?view=sql-server-ver16.

[139] Microsoft, *Cardinality Estimation (SQL Server)*, 2023. URL: https://learn.microsoft.com/en-us/sql/relational-databases/performance/cardinality-estimation-sql-server?view=sql-server-ver16.

[140] Microsoft, *Hints (Transact-SQL) - Query*, 2023. URL: https://learn.microsoft.com/en-us/sql/t-sql/queries/hints-transact-sql-query?view=sql-server-ver16.

[141] Microsoft, *Microsoft SQL Server Query Store*, 2023. URL: https://learn.microsoft.com/en-us/sql/relational-databases/performance/monitoring-performance-by-using-the-query-store?view=sql-server-ver16.

[142] Microsoft, *Microsoft SQL Server Query Store*, 2024. URL: https://learn.microsoft.com/en-us/sql/relational-databases/performance/monitoring-performance-by-using-the-query-store?view=sql-server-ver16.

[143] Microsoft, *Microsoft SQL server: Approximate Count Distinct*, 2024. URL: https://learn.microsoft.com/en-us/sql/t-sql/functions/approx-count-distinct-transact-sql?view=sql-server-ver16.

[144] Microsoft, *Microsoft SQL Server: Memory grant feedback*, 2024. URL: https://learn.microsoft.com/en-us/sql/relational-databases/performance/intelligent-query-processing-memory-grant-feedback?view=sql-server-ver16.

[145] Microsoft, *Microsoft SQL Server: Query Processing Architecture Guide*, 2024. URL: https://learn.microsoft.com/en-us/sql/relational-databases/query-processing-architecture-guide?view=sql-server-ver16.

[146] Microsoft, *Statistics: Microsoft SQL Server*, 2024. URL: https://learn.microsoft.com/en-us/sql/relational-databases/statistics/statistics?view=sql-server-ver16.

[147] Microsoft, *Automatic Plan Correction in Microsoft SQL Server*. URL: https://learn.microsoft.com/en-us/sql/relational-databases/automatic-tuning/automatic-tuning?view=sql-server-ver16.

[148] I. S. Mumick, S. J. Finkelstein, H. Pirahesh, and R. Ramakrishnan, "Magic is Relevant," in *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '90, pp. 247–258, Atlantic City, New Jersey, USA: Association for Computing Machinery, 1990. DOI: 10.1145/93597.98734.

[149] R. O. Nambiar and M. Poess, "The Making of TPC-DS," in *Proceedings of the 32nd International Conference on Very Large Data Bases*, ser. VLDB '06, pp. 1049–1058, Seoul, Korea: VLDB Endowment, 2006.

[150] V. Narasayya and S. Chaudhuri, "Cloud Data Services: Workloads, Architectures and Multi-Tenancy," *Foundations and Trends® in Databases*, vol. 10, no. 1, 2021, pp. 1–107. DOI: 10.1561/1900000060.

[151] P. Negi, Z. Wu, A. Kipf, N. Tatbul, R. Marcus, S. Madden, T. Kraska, and M. Alizadeh, "Robust Query Driven Cardinality Estimation under Changing Workloads," *Proceedings of the VLDB Endowment*, vol. 16, no. 6, 2023, pp. 1520–1533.

[152] T. Neumann, "Efficiently Compiling Efficient Query Plans for Modern Hardware," *Proceedings of the VLDB Endowment*, vol. 4, no. 9, 2011, pp. 539–550.

[153] T. Neumann and A. Kemper, *Unnesting Arbitrary Queries*, 2015.

[154] H. Q. Ngo, E. Porat, C. Ré, and A. Rudra, "Worst-case Optimal Join Algorithms," *Journal of the ACM (JACM)*, vol. 65, no. 3, 2018, pp. 1–40.

[155] F. Olken, "Random Sampling from Databases," Ph.D. dissertation, Citeseer, 1993.

[156] Oracle, *Oracle Dynamic Sampling*, 2020. URL: https://blogs.oracle.com/optimizer/post/dynamic-sampling-and-its-impact-on-the-optimizer.

[157] Oracle, *Oracle Automatic Workload Repository*, 2023. URL: https://docs.oracle.com/en/database/oracle/oracle-database/23/tgdba/awr-report-ui.html.

[158] Oracle, *Oracle Result Set Caching*, 2024. URL: https://docs.oracle.com/en/database/oracle/oracle-database/19/jjdbc/statement-and-resultset-caching.html#GUID-5D1A9E2F-F191-4FCF-994C-C1D5B143FC4F.

[159] Oracle, *Oracle SQL Tuning Guide: Histograms*, 2024. URL: https://docs.oracle.com/en/database/oracle/oracle-database/19/tgsql/histograms.html#GUID-BE10EBFC-FEFC-4530-90DF-1443D9AD9B64.

[160] Oracle, *Statistics Best Practices: Oracle*, 2024. URL: https://www.oracle.com/docs/tech/database/technical-brief-bp-for-stats-gather-19c.pdf.

[161] Oracle, *The Optimizer In Oracle Database 19c*, 2024. URL: https://www.oracle.com/technetwork/database/bi-datawarehousing/twp-optimizer-with-oracledb-19c-5324206.pdf.

[162] Oracle, *SQL Plan Management in Oracle database*. URL: https://docs.oracle.com/en-us/iaas/database-management/doc/use-spm-manage-sql-execution-plans.html.

[163] A. Pellenkoft, C. A. Galindo-Legaria, and M. L. Kersten, "The Complexity of Transformation-Based Join Enumeration," in *Proceedings of the 23rd International Conference on Very Large Data Bases*, pp. 306–315, 1997.

[164] G. Piatetsky-Shapiro and C. Connell, "Accurate Estimation of the Number of Tuples Satisfying a Condition," *ACM Sigmod Record*, vol. 14, no. 2, 1984, pp. 256–276.

[165] H. Pirahesh, T. Leung, and W. Hasan, "A Rule Engine for Query Transformation in Starburst and IBM DB2 C/S DBMS," in *Proceedings 13th International Conference on Data Engineering*, pp. 391–400, 1997. DOI: 10.1109/ICDE.1997.581945.

[166] H. Pirahesh, J. M. Hellerstein, and W. Hasan, "Extensible/Rule based Query Rewrite Optimization in Starburst," *ACM Sigmod Record*, vol. 21, no. 2, 1992, pp. 39–48.

[167] M. Poess and C. Floyd, "New TPC Benchmarks for Decision Support and Web Commerce," *SIGMOD Rec.*, vol. 29, no. 4, Dec. 2000, pp. 64–71. DOI: 10.1145/369275.369291.

[168] V. Poosala, P. J. Haas, Y. E. Ioannidis, and E. J. Shekita, "Improved Histograms for Selectivity Estimation of Range Predicates," *ACM Sigmod Record*, vol. 25, no. 2, 1996, pp. 294–305.

[169] *Postgres Query Optimizer*, 2023. URL: https://github.com/postgres/postgres/tree/master/src/backend/optimizer.

[170] *PostgreSQL pg_stats*, 2024. URL: https://www.postgresql.org/docs/current/view-pg-stats.html.

[171] *PostgreSQL: Genetic Algorithms*, 2024. URL: https://www.postgresql.org/docs/current/geqo-pg-intro.html.

[172] K. Ramachandra, K. Park, K. V. Emani, A. Halverson, C. Galindo-Legaria, and C. Cunningham, "Froid: Optimization of Imperative Programs in a Relational Database," *Proc. VLDB Endow.*, vol. 11, no. 4, Dec. 2017, pp. 432–444. DOI: 10.1145/3186728.3164140.

[173] N. Reddy and J. R. Haritsa, "Analyzing Plan Diagrams of Database Query Optimizers," in *Proceedings of the 31st International Conference on Very Large Data Bases*, ser. VLDB '05, pp. 1228–1239, Trondheim, Norway: VLDB Endowment, 2005.

[174] A. van Renen, D. Horn, P. Pfeil, K. Vaidya, W. Dong, M. Narayanaswamy, Z. Liu, G. Saxena, A. Kipf, and T. Kraska, "Why TPC is not enough: An Analysis of the Amazon Redshift fleet," *Proceedings of the VLDB Endowment*, vol. 17, no. 11, 2024, pp. 3694–3706.

[175] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe, "Efficient and extensible algorithms for multi query optimization," in *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pp. 249–260, 2000.

[176] SAP, *CREATE STATISTICS Statement in SAP Hana*, 2024. URL: https://help.sap.com/docs/SAP_HANA_PLATFORM/4fe29514fd584807ac9f2a04f6754767/20d5252d7519101493f5e662a6cda4d4.html.

[177] T. Schmidt, A. Kipf, D. Horn, G. Saxena, and T. Kraska, "Predicate caching: Query-driven Secondary Indexing for Cloud Data Warehouses," 2024.

[178] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price, "Access Path Selection in a Relational Database Management System," in *Proceedings of the 1979 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '79, pp. 23–34, Boston, Massachusetts: Association for Computing Machinery, 1979. DOI: 10.1145/582095.582099.

[179] T. K. Sellis, "Multiple-query Optimization," *ACM Transactions on Database Systems (TODS)*, vol. 13, no. 1, 1988, pp. 23–52.

[180] P. Seshadri, H. Pirahesh, and T. Leung, "Complex Query Decorrelation," in *Proceedings of the Twelfth International Conference on Data Engineering*, pp. 450–458, 1996. DOI: 10.1109/ICDE.1996.492194.

[181] S. Shankar, R. Nehme, J. Aguilar-Saborit, A. Chung, M. Elhemali, A. Halverson, E. Robinson, M. S. Subramanian, D. DeWitt, and C. Galindo-Legaria, "Query Optimization in Microsoft SQL Server PDW," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pp. 767–776, 2012.

[182] T. Siddiqui, A. Jindal, S. Qiao, H. Patel, and W. Le, "Cost Models for Big Data Query Processing: Learning, Retrofitting, and our Findings," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pp. 99–113, 2020.

[183] M. A. Soliman, L. Antova, V. Raghavan, A. El-Helw, Z. Gu, E. Shen, G. C. Caragea, C. Garcia-Alvarado, F. Rahman, M. Petropoulos, *et al.*, "Orca: a Modular Query Optimizer Architecture for Big Data," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pp. 337–348, 2014.

[184] SQLShack, *SQL Server Trivial Execution Plans*, 2021. URL: https://www.sqlshack.com/sql-server-trivial-execution-plans/.

[185] M. Steinbrunn, G. Moerkotte, and A. Kemper, "Heuristic and Randomized Optimization for the Join Ordering Problem," *The VLDB journal*, vol. 6, 1997, pp. 191–208.

[186] M. Stillger, G. M. Lohman, V. Markl, and M. Kandil, "LEO - DB2's LEarning Optimizer," in *Proceedings of the 27th International Conference on Very Large Data Bases*, ser. VLDB '01, pp. 19–28, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001.

[187] M. Stonebraker and L. A. Rowe, "The Design of Postgres," *ACM Sigmod Record*, vol. 15, no. 2, 1986, pp. 340–355.

[188] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, *et al.*, "C-store: a Column-oriented DBMS," in *Proceedings of the 31st International Conference on Very Large Data Bases*, ser. VLDB '05, Trondheim, Norway: VLDB Endowment, 2005.

[189] M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. O'Neil, *et al.*, "C-store: a Column-oriented DBMS," in *Making Databases Work: the Pragmatic Wisdom of Michael Stonebraker*, 2018, pp. 491–518.

[190] *Substrait*, 2024. URL: https://github.com/substrait-io/substrait.

[191] J. Sun and G. Li, "An End-to-end Learning-based Cost Estimator," *arXiv preprint arXiv:1906.02560*, 2019.

[192] N. Thaper, S. Guha, P. Indyk, and N. Koudas, "Dynamic Multidimensional Histograms," in *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pp. 428–439, 2002.

[193] I. Trummer, J. Wang, Z. Wei, D. Maram, S. Moseley, S. Jo, J. Antonakakis, and A. Rayabhari, "Skinnerdb: Regret-bounded Query Evaluation via Reinforcement Learning," *ACM Transactions on Database Systems (TODS)*, vol. 46, no. 3, 2021, pp. 1–45.

[194] K. Vaidya, A. Dutt, V. Narasayya, and S. Chaudhuri, "Leveraging Query Logs and Machine Learning for Parametric Query Optimization," *Proc. VLDB Endow.*, vol. 15, no. 3, Nov. 2021, pp. 401–413. DOI: 10.14778/3494124.3494126.

[195] F. M. Waas and J. M. Hellerstein, "Parallelizing Extensible Query Optimizers," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '09, pp. 871–878, Providence, Rhode Island, USA: Association for Computing Machinery, 2009. DOI: 10.1145/1559845.1559938.

[196] X. Wang, C. Qu, W. Wu, J. Wang, and Q. Zhou, "Are We Ready for Learned Cardinality Estimation?" *Proc. VLDB Endow.*, vol. 14, no. 9, May 2021, pp. 1640–1654. DOI: 10.14778/3461535.3461552.

[197] W. Wu, Y. Chi, S. Zhu, J. Tatemura, H. Hacigümüs, and J. F. Naughton, "Predicting Query Execution Time: are Optimizer Cost Models Really Unusable?" In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, IEEE, pp. 1081–1092, 2013.

[198] W. P. Yan and P.-Å. Larson, "Eager Aggregation and Lazy Aggregation," in *Proceedings of the 21th International Conference on Very Large Data Bases*, ser. VLDB '95, pp. 345–357, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995.

[199] J. Yang, S. Wu, D. Zhang, J. Dai, F. Li, and G. Chen, "Rethinking Learned Cost Models: Why Start from Scratch?" *Proceedings of the ACM on Management of Data*, vol. 1, no. 4, 2023, pp. 1–27.

[200] Z. Yang, W.-L. Chiang, S. Luan, G. Mittal, M. Luo, and I. Stoica, "Balsa: Learning a Query Optimizer Without Expert Demonstrations," in *Proceedings of the 2022 International Conference on Management of Data*, ser. SIGMOD '22, pp. 931–944, Philadelphia, PA, USA: Association for Computing Machinery, 2022. DOI: 10.1145/3514221.3517885.

[201] Z. Yang, A. Kamsetty, S. Luan, E. Liang, Y. Duan, X. Chen, and I. Stoica, "NeuroCard: One Cardinality Estimator for all Tables," *arXiv preprint arXiv:2006.08109*, 2020.

[202] Z. Yang, E. Liang, A. Kamsetty, C. Wu, Y. Duan, X. Chen, P. Abbeel, J. M. Hellerstein, S. Krishnan, and I. Stoica, "Deep Unsupervised Cardinality Estimation," *arXiv preprint arXiv:1905.04278*, 2019.

[203] M. Yannakakis, "Algorithms for Acyclic Database Schemes," in *VLDB*, vol. 81, pp. 82–94, 1981.

[204] Y. Zhao, G. Cong, J. Shi, and C. Miao, "Queryformer: A Tree Transformer Model for Query Plan Representation," *Proceedings of the VLDB Endowment*, vol. 15, no. 8, 2022, pp. 1658–1670.

[205] J. Zhu, N. Potti, S. Saurabh, and J. M. Patel, "Looking ahead makes Query Plans Robust: Making the initial case with In-memory Star Schema Data Warehouse Workloads," *Proceedings of the VLDB Endowment*, vol. 10, no. 8, 2017, pp. 889–900.

[206] R. Zhu, W. Chen, B. Ding, X. Chen, A. Pfadler, Z. Wu, and J. Zhou, "Lero: A Learning-to-rank Query Optimizer," *Proceedings of the VLDB Endowment*, vol. 16, no. 6, 2023, pp. 1466–1479.

[207] R. Zhu, Z. Wu, Y. Han, K. Zeng, A. Pfadler, Z. Qian, J. Zhou, and B. Cui, "FLAT: Fast, Lightweight and Accurate Method for Cardinality Estimation," *arXiv preprint arXiv:2011.09022*, 2020.