

Philipp Fent Technical University of Munich fent@in.tum.de

Umbra

- TUM's first DBMS acquired by Salesforce
- Rewrite from scratch
- Cutting-edge database research
- Disk-based with in-memory performance



CSRankings: Computer Science Rankings

CSRankings is a metrics-based ranking of top computer science institutions around the world. Click on a triangle (\bullet) to expand areas or institutions. Click on a name to go to a faculty member's home page. Click on a chart icon (the $\frac{1}{100}$ after a name or institution) to see the distribution of their publication areas as a bar chart \bullet . Click on a Google Scholar icon (g_0) to see publications, and click on the DBLP logo (\bullet) to go to a DBLP entry. Applying to grad school? Read this first. Do you find CSrankings useful? Sponsor CSrankings on GitHub.

2

Rank institutions in the world v by publications from 2017 v to 2023 v

~

All Areas [off | on]

AI [off | on]

- Artificial intelligence
- Computer vision
- Machine learning
- Natural language processing
- The Web & information retrieval

Systems [off | on]

- Computer architecture
- Computer networks
- Computer security
- Databases
- Design automation
- Embedded & real-time systems

Institution

- 🕨 TU Munich 🔳 📊
- 🕨 HKUST 📴 🌆
- 3 🔹 Tsinghua University 💴 📊

- 4 University of Waterloo 🙌 🌆
- 5 National University of Singapore III III
- 6 🕨 Duke University 🔤 📊
- 7 Chinese University of Hong Kong 💶 📠
- 8 🕨 Nanyang Technological University 📟 📠
- 9 🕒 Univ. of California San Diego 🎫 📊
- 10 Univ. of California Berkeley 📟 📊
- 11 🕨 Peking University 🔤 📊



Performance





Performance





🕒 UMBRA 📶

What makes Umbra fast?

- Pipelined execution
 - Keeps values in registers
 - Minimizes materialization



- Pipelined execution
- Data-centric code generation
 - Efficient code for complex expressions

```
%1 = zext i64 %int1;
                                      7ero extend to 64 bit
%2 = zext i64 %int2;
%3 = rotr i64 %2, 32;
                                              Rotate right
%v = or i64 %1, %3;
                                     Combine int1 and int2
%5 = crc32 i64 6763793487589347598, %v;
                                       First crc32
%6 = crc32 i64 4593845798347983834, %v;
                                              Second crc32
%7 = rotr i64 %6, 32;
                                         Shift second part
%8 = xor i64 %5, %7;
                                        Combine hash parts
%hash = mul i64 %8, 11400714819323198485;
                                                 Mix parts
```

🕒 UMBRA

- Pipelined execution
- Data-centric code generation R
- Fully parallel algorithms
 - Allows scaling
 - Benefits from new hardware



UMBRA

UMBRA

- Pipelined execution
- Data-centric code generation
- Fully parallel algorithms
- State-of-the-art query optimizer

- Pipelined execution
- Data-centric code generation
- Fully parallel algorithms
- State-of-the-art query optimizer

Research system with all custom advanced parts

() UMBRA

We're commercializing soon!

Query Optimization

- PostgreSQL grammar
- Parsed into relational algebra
 - Example: TPC-H Q17
 - https://umbra-db.com/interface/





Query Optimization

- PostgreSQL grammar
- Parsed into relational algebra
- Optimizer passes over algebra

- 1: Unoptimized Plan
- 2: Expression Simplification

3: Unnesting

4: Predicate Pushdown

5: Initial Join Tree

6: Sideway Information Passing

7: Operator Reordering

8: Early Probing

9: Common Subtree Elimination

10: Physical Operator Mapping

🕒 UMBRA

Query Optimization

- PostgreSQL grammar
- Parsed into relational algebra
- Optimizer passes over algebra

1: Unoptimized Plan

2: Expression Simplification

3: Unnesting

4: Predicate Pushdown

5: Initial Join Tree

8: Early Probing

6: Sideway Information Passing

9: Common Subtree Elimination

10: Physical Operator Mapping

7: Operator Reordering

Cost-based Optimization Rule-based Canonicalization

🕒 UMBRA T

Expression Simplification

- Fold constants
- Canonicalize expressions

• Execute in evaluation engine



Query Unnesting & Decorrelation

Unnesting Arbitrary Queries

Unnesting Arbitrary Queries

Thomas Neumann and Alfons Kemper Technische Universität München Munich, Germany neumann@in.tum.de, kemper@in.tum.de

Abstract: SQL-99 allows for nested subqueries at nearly all places within a query. From a user's point of view, nested queries can greatly simplify the formulation of complex queries. However, nested queries that are correlated with the outer queries frequently lead to dependent joins with nested loops evaluations and thus poor performance.

Existing systems therefore use a number of barvistics to unnext these queries; i.e., de-correctat them. These unnexting techniques can gravity speed up query processing, but are usually limited to certain classes of queries. To the best of our knowledge ne existing system can de-correlate queries in the general case. We present a generic approach for unnexting arbitrary queries. As a result, the de-correlated queries allow for much simple and much more efficient query evaluation.

1 Introduction

Subqueries are frequently used in SQL queries to simplify query formulation. Consider for our running examples the following schema:

students: {[id, name, major, year, ...]}

• exams: {[sid, course, curriculum, date, ...]}

Then the following is a nested query to find for each student the best exams (according to the German grading system where lower numbers are better):

Conceptually, for each student, exam pair (s,e) it determines, in the subquery, whether or not this particular exam e has the best grade of all exams of this particular student s.

From a performance point of view the query is not so nice, as the subquery has to be reevaluated for every student, exam pair. From a technical perspective the query contains a

383

DuckDB

Documentation ~ Blog

Blog

2023-05-26 Mark Raasveldt

Correlated Subqueries in SQL

Subqueries in SQL are a powerful abstraction that allow simple queries to be used as composable building blocks. They allow you to break down complex problems into smaller parts, and subsequently make it easier to write, understand and maintain large and complex queries.

DuckDB uses a state-of-the-art subquery decorrelation optimizer that allows subqueries to be executed very efficiently. As a result, users can freely use subqueries to create expressive queries without having to worry about manually rewriting subqueries into joins. For more information, skip to the Performance section.

Types of Subqueries

SQL subqueries exist in two main forms: subqueries as expressions and subqueries as tables. Subqueries that are used as expressions can be used in the SELECT or INHERE clauses. Subqueries that are used as tables can be used in the FROM clause. In this blog post we will focus on subqueries used as expressions. A future blog post will discuss subqueries as tables.

Subqueries as expressions exist in three forms.

- Scalar subgueries
- EXISTS
- IN / ANY / ALL

All of the subqueries can be either correlated or uncorrelated. An uncorrelated subquery is a query that is independent from the outer query. A correlated subquery is a subquery that contains expressions from the outer query. Correlated subqueries can be seen as parameterized subqueries.

Query Unnesting

- Unnesting Arbitrary Queries
 - O(n²)





Query Unnesting

- Unnesting Arbitrary Queries
 - O(n²)



Query Unnesting

- Unnesting Arbitrary Queries
 - O(n²) -> O(n)
 - Huge improvement





- Place predicates at scan
- Propagate & fold constants



- Place predicates at scan
- Propagate & fold constants



- Place predicates at scan
- Propagate & fold constants



- Place predicates at scan
- Propagate & fold constants





Initial Join Tree

- Push joins through aggregates
- Expand transitive join conditions

```
c_nationkey = s_nationkey
and s_nationkey = n_nationkey
==
```

```
c_nationkey = s_nationkey
and s_nationkey = n_nationkey
and c_nationkey = n_nationkey
```



Initial Join Tree

- Push joins through aggregates
- Expand transitive join conditions
- Drop unnecessary joins

```
select sum(o_totalprice)
from customer, orders
where c_custkey = o_custkey
```

```
==
```

select sum(o_totalprice)
from orders



Cost-Based Optimization

• Heuristics vs. statistics

Cost-Based Optimization

- Heuristics vs. statistics
- Statistics in Umbra:
 - \circ Samples
 - Distinct counts
 - Numerical statistics (mean, variance) for aggregates

UMBRA

- Functional dependencies
- \Rightarrow Estimate execution cost



- Maintain uniform reservoir sample
- Evaluate scan predicates σ on sample
- Execute in evaluation engine
- Surprisingly accurate
 - 1024 tuples ~ 0.1% error

```
select count(*)
  from lineitem
  where l_commitdate < l_receiptdate
   and l_shipdate < l_commitdate</pre>
```



for l in lineitem: if not l_shipdate < l_commitdate: continue -- 51% taken if not l_commitdate < l_receiptdate: continue -- 75% taken

counter++

Variant (A) : Separate branches

for l in lineitem: if not l_commitdate < l_receiptdate: continue -- 37% taken if not l_shipdate < l_commitdate: continue -- 81% taken

for l in lineitem:
 if not (l_shipdate < l_commitdate
 and l_commitdate < l_receiptdate):
 continue -- 88% taken</pre>

counter++

Variant (B) : Separate branches

counter++

Variant \bigcirc : Combined branch



for l in lineitem: if not l_shipdate < l_commitdate: continue -- 51% taken if not l_commitdate < l_receiptdate: continue -- 75% taken for l in lineitem: if not l_commitdate < l_receiptdate: continue -- 37% taken if not l_shipdate < l_commitdate: continue -- 81% taken

for l in lineitem:
 if not (l_shipdate < l_commitdate
 and l_commitdate < l_receiptdate):
 continue -- 88% taken</pre>

counter++

Variant A : Separate branches

counter++

Variant (B): Separate branches

counter++

Variant \bigcirc : Combined branch

Variant	branch-misses	instructions	loads	exec. time
A	0.63 / tpl	7.62 / tpl	2.85 / tpl	18.4 ms
B	0.58 / tpl	7.91 / tpl	3.00 / tpl	17.7 ms
\bigcirc	0.13 / tpl	11.67 / tpl	3.37 / tpl	12.7 ms



- Estimate (correlated) predicates with confidence
- Any combination of predicates
- Tricky when 0 / 1024 tuples qualify
- Can do better for conjunctions

Research Data Management Track Paper

SIGMOD '21, June 20-25, 2021, Virtual Event, China

Small Selectivities Matter: Lifting the Burden of Empty Samples

Axel Hertzschuch Fechnische Universität Dresden xel.hertzschuch@tu-dresden.de	Guido Moerkotte University of Mannheim moerkotte@uni-mannheim.de	Wolfgang Lehner Technische Universität Dresden wolfgang.lehner@tu-dresden.de	
Norman May	Florian Wolf	Lars Fricke	
SAP SE	SAP SE	SAP SE	
norman.may@sap.com	florian.wolf01@sap.com	lars.fricke@sap.com	

ABSTRACT

Every year more and more advanced approaches to cardinality estimation are published, using learned models or other data and workload specific synopses. In contrast, the majority of commercial in-memory systems still relies on sampling. It is arguably the most general and easiest estimator to implement. While most methods do not seem to improve much over sampling-based estimators in the presence of non-selective queries, sampling struggles with highly selective queries due to limitations of the sample size. Especially in situations where no sample tuple qualifies, optimizers fall back to basic heuristics that ignore attribute correlations and lead to large estimation errors. In this work, we present a novel approach, dealing with these 0-Tuple Situations. It is ready to use in any DBMS capable of sampling, showing a negligible impact on optimization time. Our experiments on real world and synthetic data sets demonstrate up to two orders of magnitude reduced estimation errors. Enumerating single filter predicates according to our estimates reveals 1.3 to 1.8 times faster query responses for complex filters.

ACM Reference Format:

Axel Hertzschuch, Guido Moerkotte, Wolfgang Lehner, Norman May, Florian Wolf, and Lars Fricke. 2021. Small Selectivities Matter: Lifting the Burden of Empty Samples. In Proceedings of the 2021 International Conference on Management of Data (SGMOD '21), June 18–27, 2021, Virtual Event, China. ACM. New York, NY, USA. 13 pages. https://doi.org/10.1145/3448005.3452805

1 INTRODUCTION

Good cardinality estimates guide query optimizers towards decent execution plans and lower the risk of disastrous plans [32, 23], Although many approaches were published on cardinality estimation, e.g., using histograms [18], sampling [11,] er machine learning [13], it is still considered a grand challenge [28]. Especially analytical workdoads remain challenging as they often comprise a multitude of correlated filter predicates. The comprehensive analysis of 60k realworld Bit data repositories by Vogedsegnast et al. [45] underlines the importance of filter operations and reveals: Most data is stored in string format, which enables athritary complex expressions.



Figure 1: Relative number of queries over tables with at least 1M tuples that lead to empty samples (0-TS) with regard to the number of filter predicates (atoms) and the sample size.

Sampling is an ad-hoc approach that captures correlations among arbitrary numbers and types of predicates. Therefore, it is commonly used in commercial systems [25, 26, 36, 40] and has been combined with histograms [35] and machine learning [23, 47]. However, it is not a panacea. Although sampling might be reasonably fast for in-memory systems due to the efficient random access [17], the number of sample tuples often is very limited. Traditionally, we randomly draw a fixed number of tuples from a table and divide the number of qualifying sample tuples by the total number of sample tuples. Instead of drawing the sample at query time. some approaches exploit materialized views [24] or use reservoir sampling [7, 44]. Given a sufficient number of qualifying tuples. these sample-based estimates are precise and give probabilistic error guarantees [32]. However, complex predicates frequently lead to situations where no sample tuple qualifies. According to Kinf et al. [22] we call these 0-Tuple Situations (0-TS). To assess the frequency at which 0-TS occur, we analyze the Public Bi Benchmark [2], a real-world, user-generated workload, Considering base tables with at least 1M tuples. Figure 1 illustrates the relative number of queries that result in 0-TS when using two standard sized random samples. Interestingly, and contrary to the intuition of being a corner case, this analysis of a real-life workload reveals that up to 72% of the queries with complex filters lead to empty samples. In these situations, query optimizers rely on basic heuristics, e.g., using Attribute Value Independence (AVI), that lead to large estimation errors and potentially poor execution plans [33, 37]. To illustrate this deficiency, suppose we sample from a table containing brands, models and colors of cars. Even if no sample tuple qualifies for a given filter, there is little justification to assume independence between all attributes as the model usually determines the brand. Surprisingly, no previous work we are aware of considers correlations in 0-TS. This paper therefore presents a novel approach that - given a sample - derives more precise selectivity estimates

Permission to make digital or band cogien of all arg att of this work for personal or automorum usin ig gatarowiden for periodel material organs are at made of distributed control of the second seco



- Calculate matches-bitsets
- Combine them to optimize ordering

```
• TPC-H Q12:
```

```
where l_shipmode in ('MAIL', 'SHIP')
and l_commitdate < l_receiptdate
and l_shipdate < l_commitdate
and l_receiptdate between date '1994-01-01'
and date '1994-12-31'</pre>
```

0100'0011'1010'0100'1110'1011'1011'1100'1010'1010'1010'1011'0000'1011'0011'1100'0000 & 0000'1111'0000'1111'0000'1111'0000'1111'0000'1111'0000'1111'0000'1111'0000'1111'0000'1111' & 1111'0000'1111'0000'1111'0000'1111'0000'1111'0000'1111'0000'1111'0000'1111'0000'1111'0000'1111'0000'0001 & 1010'0110'1110'1100'0011'0111'0101'0110'1111'1001'1101'1100'0011'1000'0001



Early Execution

- Size of sample > table size
- Allows a third round of constant propagation
 - Especially for small fact tables

```
select r_regionkey
  from region
  where r_name = 'Europe'
  ==
  select 3
```

Join Ordering

- Hash Joins rule
 - Indexes don't allow bushy plans -> less useful

Bernhard Radke

racke@in.tum.de

normalized cost flog scale Figure 1: Normalized Cost Distribution of Random Pla a Data-Warehouse-Style Query with 59 Relations

lead to (generated) ad-hoc queries that can easily touch a hundred lead to (generated) ad-loce queries that care easily tooch a handred rulations, and a database system must be able to bandle these, ion. Toren moderately since queries with, e.g., 50 relations are far be-yout what can be optimized carefully. In such cases, optimizers have to sacrifice optimizity and employ heuristics to keep optimization time reasonable. Finare 1 above the darifution of the costs neema-

Adaptive Optimization of Very Large Join Queries

Thomas Neumann

Technische Universität Mänchen neumann@in.tum.de

ABSTRACT The or of business indigence tasks and other means is granning means had to prove variety as its aim of spin squares. While the spin strength is t

rear-optimal execution plans for large classes of queries. In addi-

to scale join cedering algorithms to these entremely large queries. Extensive experiments with over 10 different approaches show that

f query sizes, and produces optimal or near-optima

the new adaptive approach proposed here performs excellent over

ABSTRACT

Join Ordering

- Hash Joins rule
 - Indexes don't allow bushy plans -> less useful Ο



Join Ordering

- Hash Joins rule
 - Indexes don't allow bushy plans -> less useful
- Distinct count estimates with Pat Sellinger's equations
- HyperLogLog intersections
- Mean & stddev approximations for l_quantity < 0.2 * avg(l_quantity)

Early Probing

- Semijoin reduction
- Reuses existing hash tables
- Can use bloom filters if beneficial



Physical Optimization

- Indexes
- Worst-case optimal join

Adopting Worst-Case Optimal Joins in **Relational Database Systems**

Michael Freitag, Maximilian Bandle, Tobias Schmidt, Alfons Kemper, Thomas Neumann (freitagm, bandle, tobias.schmidt, kemper, neumann)@in.tum.de

ABSTRACT

Worst-case optimal join algorithms are attractive from a theoretical point of view, as they offer asymptotically bet-ter runtime than binary joins on ortain types of queries. In particular, they avoid enumerating large intermediate reoverhead in practice, primarily since they rely on suitable ordered index structures on their input. Systems that sup-port worst-case optimal joins often focus on a specific prob-lem domain, such as read-only graph analytic queries, where tion approach for worst-case optimal joins that is practical within general-purpose relational database management systems supporting both hybrid transactional and analyt-ical workloads. The key component of our approach is a lies only on data structures that can be built efficiently dur-ing query execution. Furthermore, we implement a hybrid query optimizer that intelligently and transparently combines both binary and multi-way joins within the same query plan. We demonstrate that our approach far outperforms existing systems when worst-case optimal joins are beneficial while sacrificing no performance when they are not.

PVLDB Reference Format: Michael Freitag, Maximilian Beadle, Tohias Schmidt, Alfons Kem-per, and Thomas Neumann. Adopting West-Case Optimal Joins in Relational Database Systems. *PVLDB*, 13(11): 1891-1994,

1. INTRODUCTION

The vast majority of traditional relational database management systems (RDBMS) relies on binary joins to pro-cess queries that involve more than one relation, since they ing to decades of optimization and fine-tuning, they offer great flexibility and excellent performance on a wide range

This work is licensed under the Creative Commons Attribution-NetCommercial-NetGerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-ne-nd4/07. For any use beyond these corrected by this license, obtain permission by emailing into Wyddhorg. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endownest. licensed to the VLDB Endowment. Proceedings of the VLDB Endowment, Vol. 13, No. 11 ISSN 2150-8097. DOI: https://doi.org/10.14778/3407790.3407797

of workloads. Nevertheless, it is well-known that there are pathological cases in which any binary join plan exhibits suboptimal performance [10,19,30]. The main shortcoming of binary joins is the generation of intermediate results that can become much larger than the actual query result [46]. Unfortunately, this situation is generally unavoidable in complex analytical settings where joins between non-key attributes are commonplace. For instance, a conceivable ouer on the TPCH schema would be to look for parts within the same order that could have been delivered by the same supplier. Answering this query involves a self-join of lineiter and two non-key joins between lineiten and partrupp, all of which generate large intermediate results 16. Self-joins that incur this issue are also prevalent in graph anajoin that izerr this issue are also prevalent in graph ana-tyric queries such as searching for triangle pattern within a graph $\frac{1}{2}$. On such queries, traditional RDDMS that employ ionary yina plass frequently achieved instatoms performance or even hit to produce any result at all $\frac{1}{2}\frac{1}{2}\frac{1}{2}\frac{1}{2}\frac{1}{2}\frac{1}{2}}$. Consequently, there has been a large-standing interest of works on plans that avoid emmerrising any potentially en-gradement of the state of the state of the state of the state of advances recording random the state of the state of the potential to tight bounds on the works case size of the neuro-perional to tight bounds on the works are size of the neuroportional to tight bounds on the worst-case size of the query result 04546540.). As they can guarantee better asymptotic runtime complexity than binary join plans in the persence of growing intermediate results, they have the potential to greatly improve the robustness of relational database sys-tems. However, existing implementations of worst-case op-timal joins have several shortcomings which have impeded

UMBRA

2020. DOI: https://doi.org/10.14778/3407790.3407797

timal joins here several abstromaging which have imposed to adoption which merge productions proteins of loc. The adoption which merge production proteins of loc. As a stributes that can particle in a join which estimates and attributes that can particle in a join which estimates the stributes of the stributes of the stributes that are excellenged with one contribution of the stributes that are term does support mutable data. For each other ways the states are stributed as the stributes of the stributes that and a stribute stributes are stributed as the stributes of the stributes of the stributes of the stributes of the stributes and a stribute stributes of the stributes of the stributes are joint these are no growing intermediate reads the truth are growing the stributes of the stributes of the stributes of the stributes are stributed as a protect and the stributes of the str

nerrose RDBMS measing (1) an optimizer that only intro purpose RDDDDS requires (1) an optimizer that only intro-duces a multi-way join if there is a tangible benefit in doing so, and (2) performant indexes structures that can be built efficiently on-the-flu and do not have to be persisted to disk.

Physical Optimization

- Indexes
- Worst-case optimal join
- Groupjoin

Adopting Worst-Case Optimal Joins in **Relational Database Systems**

Michael Freitag, Maximilian Bandle, Tobias Schmidt, Alfons Kemper, Thomas Neumann {freitagm, bandle, tobias.schmidt, kemper, neumann}@in.tum.de

ABSTRACT

Worst-case optimal join algorithms are attractive from a theoretical point of view, as they offer asymptotically bet-ter runtime than binary joins on certain types of queries In particular, they avoid commerating large intermediate reoverhead in practice, primarily since they rely on suitable ordered index structures on their input. Systems that supordered index structures on their input. Systems that sup-port worst-case optimal joins often focus on a specific prob-lem domain, such as read-only graph analytic queries, where tion approach for worst-case optimal joins that is practical within general-purpose relational database management systems supporting both hybrid transactional and analyt-ical workloads. The key component of our approach is a lies only on data structures that can be built efficiently dur-ing query execution. Furthermore, we implement a hybrid query optimizer that intelligently and transparently combines both binary and multi-way joins within the same query plan. We demonstrate that our approach far outperforms existing systems when worst-case optimal joins are beneficial while sacrificing no performance when they are not-

PVLDB Reference Format: Michael Freitag, Maximilian Beadle, Tobias Schmidt, Allons Kem-per, and Thomas Neumann. Adopting Worst-Case Optimal Joins in Relational Database Systems. *PVLDB*, 13(11): 1891-1894.

A Practical Approach to Groupjoin and Nested Aggregates

Philipp Fent

Technische Universität München

fent@in.tum.de

Groupjoins, the combined execution of a join and a subsequent

group by, are common in analytical queries, and occur in about 10

of the queries in TPC-H and TPC-DS. While they were originally invented to improve performance, efficient parallel execution of groupioins can be limited by contention, which limits their useful-

tess in a many-core system. Having an efficient implementation

of groupioins is highly desirable, as groupioins are not only used

to fuse group by and join but are also introduced by the unnesting

Thomas Neumann Technische Universität Müncher neumann@in.tum.de

UMBRA

H

Figure 1: Missing components for practical groupjoins. Our improvements to estimation and parallel execution enable efficient evaluation of queries with nested aggregates

The primary reason to use a groupjoin, is its performance. We spend less time building hash tables, use less memory, and improve the responsiveness of this query. However, groupioins are also more capable than regular group-bys, as we can create the groups explicitly. Consider the following nested query, with subtly different

SELECT cust.id. cnt. s

FROM customer cust, (SELECT COUNT(+) AS cnt. SUM(s.value) as s FROM sales s

WHERE cust.id = s.c.id

Here, nested the query calculates a COUNT(*) over the inner table, which evaluates to zero when there are no join partners. Answerin that query without nested-loop evaluation of the inner query i tricky, as a regular join plus group-by will produce wrong result for empty subqueries, which is known as the COUNT bug [44]. A groupioin directly supports such queries by evaluating the static aggregate for the nested side of the join, taking the groups from the other side.

Despite their benefits, groupjoins are not widely in use. We identify two problems and propose solutions that make groupjoins more practical: First, existing algorithms for groupjoins do not scale well for rarallel execution. Since the groupioin hash table contains shared aggregation state, parallel updates of these need synchronization, and can cause beavy memory contention. Furthermore current estimation techniques deal poorly with results of groupjoins from unnested aggregates.

The unnesting of inner aggregation subqueries is very prof table, since it eliminates nested-loops evaluation and improves the asymptotic complexity of the query. However, this causes the aggregates to be part of a bigger query tree, mangled between joins, predicates and other relational operators. Operv optimiza tion, specifically join ordering, depends on the quality of cardinality

ISSN 2150-8097. DOI: https://doi.org/10.14778/3407790.3407797

2020. DOI: https://doi.org/10.14778/3407700.3407707

1. INTRODUCTION

The vast majority of traditional relational database managreent systems (RDBMS) relies on binary joins to pro-cess queries that involve more than one relation, since they ing to decades of optimization and fine-tuning, they offer great flexibility and excellent performance on a wide range

This work is licensed under the Creative Commons Attribution NetCommercial-NoDerivatives 4.0 International License. To view a core into the video or a construction of the stream of the second of the stream of the second of the seco

can become much larger than the actual q Unfortunately, this situation is general complex analytical settings where joins bet tributes are commonplace. For instance, a on the TPCH schema would be to look for same order that could have been delivered plier. Answering this query involves a set and two non-key joins between lineiten all of which generate large intermediate re joins that incur this issue are also prevale lytic queries such as searching for triangle p graph 3. On such queries, traditional RDI binary join plans frequently exhibit disastr or even fail to produce any result at all

of workloads. Nevertheless, it is well-know

pathological cases in which any binary jo suboptimal performance 10 19 30. The m of binary joins is the generation of intermed

or even fail to produce any result at all 2 Consequently, there has been a long-sta-scalif-way joins that avoid enumerating an ploding intermediate results [10][19][30]. cal advances recently enabled the develops optional multi-way join algorithms which I component of the query optimizer to avoid nested-loops evalu ation of aggregates. Furthermore, the query optimizer needs be able to reason over the result of appreciation in order to schedule it correctly. Traditional selectivity and cardinality estimation quickly reach their limits when faced with computed columns from portional to tight bounds on the worst-case result 9454654. As they can guarantee b runtime complexity than binary join plans of growing intermediate results, they have nested aggregates, which leads to poor cost estimations and thus, subcotimal mery plans. In this paper, we present techniques to efficiently estimate, plan and execute groupioins and nested aggregates. We propose two novel techniques, aggregate estimates to predict the result distribut tion of appreciates and ografiel proution execution for a scalable timal joins have several shortcomings wh execution of groupjoins. The resulting system has significantly bet First, they require suitable indexes on of attributes that can partake in a join ter estimates and a contention-free evaluation of groupjoins, which can speed up some TPC-H queries up to a factor of 2. enormous storage and maintenance overh general-purpose RDBMS must support in whereas worst-case optimal systems like PVLDB Reference Format: Philipp Fent and Thomas Neumann. A Practical Approach to Groupjoin LevelHeaded rely on specialized readand Nested Aggregates. PVLDB, 14(11): 2383 - 2396, 2021. quire expensive precomputation 23. The tem does support mutable data, but can be nitude slower than such read-optimized

doi:10.14778/3476249.3476285 1 INTRODUCTION nally, multi-way joins are commonly nut Joins and aggregations are the backbone of query engines. A common query pattern, which we observe in many benchmarks [7, 45] We thus argue that an implementation and industry applications [58], is a join with grouped aggregation nermose RDBMS measines (1) an optimiz duces a multi-way join if there is a tangible so, and (2) performant indexes structures t on the same key SELECT cust.id. COUNT(+). SUM(s.value) efficiently on-the-flu and do not have to be FROM customer cust, sales s WHERE cust.id = s.c_id

ABSTRACT

GROUP BY cust.id

In a traditional implementation, we answer the query by building two hash tables on the same key, one for the hash join and one for the group-by. However, we can speed up this query by reusing the join's hash table to also store the aggregate values. This combined execution of join and group-by is called a groupioin [42].

This work is likensed under the Creative Common MY-NC-ND 4.0 https:// likense.Wish.Bhys.Greativerements.org/likenses/by-nc.nd/cd/2 to view a copy of this license.Far are use boyered that accounted by this license, bara start baracteristics by emailing inder/dd/ang, Copyraph is held by the owner/stather(b). Publication rights licensed to the VLDB Endorment, Vol. 14, No. 11 ISSN 2156-8097. doi:10.1077/0115045-315828

Physical Optimization

- Indexes
- Worst-case optimal join
- Groupjoin
- Range join
- Join micro-optimizations
 - Multiset semantics \cap
 - Allocation sizes \bigcirc

Adopting Worst-Case Optimal Joins in **Relational Database Systems**

Michael Freitag, Maximilian Bandle, Tobias Schmidt, Alfons Kemper, Thomas Neumann (freitagm, bandle, tobias.schmidt, kemper, neumann)@in.tum.de

ABSTRACT

Worst-case optimal join algorithms are attractive from a theoretical point of view, as they offer asymptotically bet-ter runtime than binary joins on certain types of queries In particular, they avoid enumerating large intermediate reoverhead in practice, primarily since they rely on suitable ordered index structures on their input. Systems that sup-port worst-case optimal joins often focus on a specific prob-lem domain, such as read-only graph analytic queries, where tion approach for worst-case optimal joins that is pract cal within general-purpose relational database management systems supporting both hybrid transactional and analyt-ical workloads. The key component of our approach is a query optimizer that intelligently and transparently con

Consequently, there has been a long-st wall-seq joins that avoid enumerating ploding intermediate results 10 19 50. A Scalable and Generic Approach to Range Joins

Thomas Neumann Technical University of Munich Technical University of Munich neumann@in.tum.de

of workloads. Nevertheless, it is well-know

pathological cases in which any binary jo suboptimal performance 10 19 30. The m of binary joins is the generation of intermed

can become much larger than the actual q Unfortunately, this situation is general complex analytical settings where joins bet

tributes are commonplace. For instance, a

on the TPCH schema would be to look for same order that could have been delivered

plier. Answering this query involves a set

and two non-key joins between lineiten all of which generate large intermediate re joins that incur this issue are also prevale

lytic queries such as searching for triangle p graph 3. On such queries, traditional RDI binary join plans frequently exhibit disastr

ABSTRACT

PVI Mid per, in R

Analytical database systems provide great insights into large datasets and are an excellent tool for data exploration and analysis. A central pillar of query processing is the efficient evaluation of equi-joins typically with linear-time algorithms (e.g. hash joins). However, for many use-cases with location and temporal data, non-equi joins, like range joins, occur in queries. Without optimizations, this typically results in nested loop evaluation with quadratic complexity. This leads to unacceptable query execution times. Different mitigations have been proposed in the past, like partitioning or sorting the data. While these allow for handling certain classes of querie they tend to be restricted in the kind of overies they can support And, perhaps even more importantly, they do not play nice with additional equality predicates that typically occur within a query and that have to be considered, too. In this work, we present a kd-tree-based, multi-dimension range

Maximilian Reif

reif@in.tum.de

join that supports a very wide range of queries, and that can exploit additional equality constraints. This approach allows us to handle large classes of queries very efficiently, with negligible memory overhead, and it is suitable as a general-purpose solution for range operies in database systems. The join algorithm is fully parallel both during the build and the probe phase, and scales to large problem instances and high core counts. We demonstrate the feasibility of this approach by integrating it into our database system Umbra and performing extensive ex-

periments with both large real world data sets and with synthetic benchmarks used for sensitivity analysis. In our experiments, it outperforms hand-tuned Spark code and all other database system that we have tested

PVLDB Reference Format:

Maximilian Beif and Thomas Neumann, A Scalable and Generic Approach to Range Joins. PVLDB, 15(11): 3018 - 3030, 2022. doi:10.14778/3551793.3551849

PVI DB Artifact Availability The source code, data, and/or other artifacts have been made available at https://eitlab.db.in.tum.de/max.reif/rangeioin-reproducibility.

1 INTRODUCTION

Over the last years, we observed two major trends in data processing: The amount of data collected is vastly growing, and data analysis techniques are becoming more and more refined. Database systems provide an excellent base for managing these very large

This work is locused under the Creative Commons IF-NC-ND 4.0 International Learnes. Viral https://creativecommon.org/learnes/by-nc-rel/40.1 to-view a copy of this literate. For any use beyond these covered by this literate, a binary of the second to the VLDB International International to the VLDB International Common Second Second Second Second Proceedings of the VLDB International VLDB International Second Sec

A Practical Approach to Groupjoin and Nested Aggregates

Philipp Fent

Technische Universität München

fent@in.tum.de

Groupjoins, the combined execution of a join and a subsequent

group by, are common in analytical queries, and occur in about 10

of the queries in TPC-H and TPC-DS. While they were originally invented to improve performance, efficient parallel execution of

groupioins can be limited by contention, which limits their useful

of groupioins is highly desirable, as groupioins are not only used

to fuse group by and join but are also introduced by the unnesting

component of the query optimizer to avoid nested-loops evalu ation of aggregates. Furthermore, the query optimizer needs be

able to reason over the result of appreciation in order to sched-

y and cardinality estimation:

with computed columns from cost estimations and thus

s to efficiently estimate, plan

argregates. We propose two

to predict the result distribu

sinin execution for a scalable

tical Approach to Groupjoin

one of query engines. A com

s in many benchmarks [7, 45]

in with grouped argregation

Jun this overy by reasing the

rerate values. This combines

seen BY-NC-ND 4.0 International mea-by-ne-nd/4.0, to view a copy of y that license, abtain permission by e owner/sather(s). Publication rights

11 ISSN 2150-9097.

to a factor of 2.

3 - 2396, 2021.

system has significantly betuation of groupioins which

tess in a many-core system. Having an efficient implementation

ABSTRACT

Thomas Neumann Technische Universität Müncher neumann@in.tum.de

🕒 UMBRA



Figure 1: Missing components for practical groupjoins. Ou rovements to estimation and parallel execution enable efficient evaluation of queries with nested aggregates

The primary reason to use a groupjoin, is its performance. We spend less time building hash tables, use less memory, and improve the responsiveness of this query. However, groupioins are also more capable than regular group-bys, as we can create the groups explicitly. Consider the following nested query, with subtly different

SELECT cust.id. cnt. s FROM customer cust, (

SELECT COUNT(*) AS cnt. SUM(s.value) as s

FROM sales s WHERE cust.id = s.c.id

Here, nested the query calculates a COUNT(*) over the inner table which evaluates to zero when there are no join partners. Answerin that query without nested-loop evaluation of the inner query i tricky, as a regular join plus group-by will produce wrong result for empty subqueries, which is known as the COUNT bug [44]. A groupioin directly supports such queries by evaluating the statiaggregate for the nested side of the join, taking the groups from the other side.

Despite their benefits, groupjoins are not widely in use. We identify two problems and propose solutions that make groupjoins more practical: First, existing algorithms for groupjoins do not scale well for rarallel execution. Since the groupioin hash table contains shared aggregation state, parallel updates of these need synchronization, and can cause heavy memory contention. Furthermore current estimation techniques deal poorly with results of groupjoins from unnested aggregates.

The unnesting of inner aggregation subqueries is very protable, since it eliminates nested-loops evaluation and improves the asymptotic complexity of the query. However, this causes the aggregates to be part of a bigger query tree, mangled between joins, predicates and other relational operators. Onery optimiza tion, specifically join ordering, depends on the quality of cardinality

T1.dest = T2.orig and f2.takeoff between f1.landing + '45 ninutes' and f1.landing + '3 hours' order by f1.price + f2.price limit 10 In this case, the join has two join conditions. The equivalence predicate f1.dest = f2.orig and the range predicate f2.takeoff between f1.landing + '45 minutes' and f1.landing + '3 hours'. Thus, the join could be considered an equi-join with a range-residual or a range join with an additional equivalence-predicate. Other examples for range joins are: The matching of vehicle sensor data to vehicle ride (defined by a time frame) or the mapping of IP addresses to subnet [37]. Moreover, there are applications, which require the evaluation of multiple range predicates, so-called multi-dimensional range joins. Examples are: Finding return trips in taxi-ride datasets (Sec tion 6.3.3) or combining hird sightings and weather reports [23 based on location and time data. Additional equivalence predicates as in the flight example, are also very common and should be in corporated into a range join algorithm.

Figure 1: Flight routing with stop-over

datasets and provide highly tuned implementations to rapidly an

swer analytical questions. One very typical and well-understoor

challenge are joins on large amounts of data based on equivalence

A straightforward example is a flight routing search: Given a

flights from Munich to Sydney. Since no direct flights are available.

A major constraint is that we are only interested in connection

with a transit duration between 45 minutes and three hours. A

query answering this question could look like this:

f1.orig = 'MUC' and f2.dest = 'SYD' and f1.dest = f2.orig and

from flights fl, flights f2

we want to find connections with a stopover, as shown in Figure 1.

so-called range joins.

predicates. However, for many datasets (especially with temporal uswer the query by building

or sensor data) queries arise that contain joins on range predicates, for the hash join and one for

large database of flight connections, we would like to find affordable lod a groupion [42].

Recap

- Query compilation & optimization
 - Optimizer passes
 - Rule-based canonicalization
 - Cost-based optimization
- Cutting-edge research
 - Join ordering
 - Cardinality estimation
 - Integrated in a running system

- 1: Unoptimized Plan
- 2: Expression Simplification

UMBRA

- 3: Unnesting
- 4: Predicate Pushdown
- 5: Initial Join Tree
- 6: Sideway Information Passing
- 7: Operator Reordering
- 8: Early Probing
- 9: Common Subtree Elimination
- 10: Physical Operator Mapping

Conclusion

- Low latency analytical queries
- Also works excellent for transactional and graph workloads

We are commercializing

Reach out:

fent@in.tum.de

TUM Open Source Project

