

Carnegie Mellon University

OPTIMIZE!

Database Query Optimization

Optimizer Generators
ft. Exodus + Volcano

ERRATA

Creepy Request

Typo in SQL

Send Corrections: db-mistakes@cs.cmu.edu

ERRA

Creepy Request Typo in SQL

Send Corrections: db-m

[DB Mistakes] Help My Marriage Problem???

From: [Redacted]
 To: "db-mistakes@cs.cmu.edu" <db-mistakes@cs.cmu.edu>
 Sender: Db-mistakes <db-mistakes-bounces@mailman.srv.cs.cmu.edu>
 List-Id: <db-mistakes.mailman.srv.cs.cmu.edu>
 Date: 1/22/25 1:01 PM
 Spam Status: Spamassassin

I have lots of problems with my wife. My marriage is unraveling. I don't know what to do. I watch your lectures all the time and you've always struck me as someone who understands the intricacies of human connection. I need your help to save what's left of my relationship. I know this is unconventional, but I invite you to move in with us for a while. This is not meant to be a creepy time thing. You will sleep in your own room. It will bring balance to our fractured household. Please consider this.

[Redacted]
 [Redacted]
 [Redacted]
 [Redacted]

FRR

[DB Mistakes] Typo in Lecture #03 (Starbust) slide (Query optimization)



From: [Redacted]
 To: db-mistakes@cs.cmu.edu
 Sender: [Db-mistakes <db-mistakes-bounces@mailman.srv.cs.cmu.edu>](mailto:Db-mistakes<db-mistakes-bounces@mailman.srv.cs.cmu.edu>)
 List-Id: [<db-mistakes@mailman.srv.cs.cmu.edu>](mailto:db-mistakes@mailman.srv.cs.cmu.edu)
 Date: 1/26/25 2:35 PM
 Spam Status: Spamassassin

Hi,

It seems to me that in slide #18 there is a typo in the SQL shown:

```
SELECT DISTINCT q1.partno, q1.descr, q2.supplyno
FROM inventory AS q1, quotations AS q2
WHERE q1.partno = q2.partno
AND q1.descr = 'engine'
AND *q1.price * <= ALL(
  SELECT q3.price
  FROM quotations AS q3
  WHERE q2.partno = q3.partno );
```

"*q1*.price <= ALL(...)" should be "*q2*.price <= ALL(...)", i.e. finding the quotation with the lowest price among all quotations for the 'engine'.

Thank you for making the lectures available for us outsiders. It is an absolute blast for me.

Best wishes,

[Redacted]

[DB Mistakes] Help My Marriage Problem???

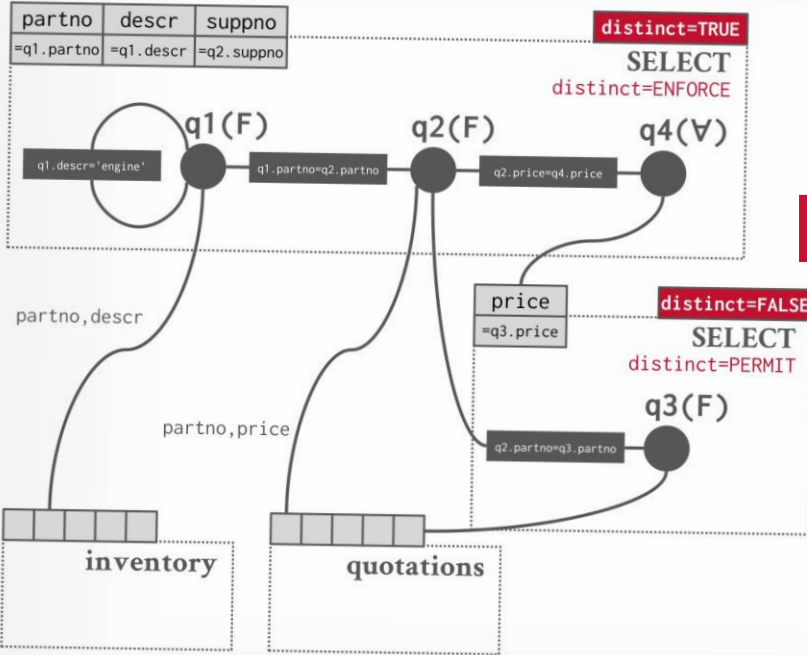


From: [Redacted]
 To: ["db-mistakes@cs.cmu.edu" <db-mistakes@cs.cmu.edu>](mailto:db-mistakes@cs.cmu.edu)
 Sender: [Db-mistakes <db-mistakes-bounces@mailman.srv.cs.cmu.edu>](mailto:Db-mistakes<db-mistakes-bounces@mailman.srv.cs.cmu.edu>)
 List-Id: [<db-mistakes@mailman.srv.cs.cmu.edu>](mailto:db-mistakes@mailman.srv.cs.cmu.edu)
 Date: 1/22/25 1:01 PM
 Spam Status: Spamassassin

I've lots of problems with my wife. My marriage is unraveling. I don't know what to do. I watch your lectures all the time and you've always helped me as someone who understands the intricacies of human connection. I would like your help to save what's left of my relationship. I know this is unconventional, but I invite you to move in with us for a while. This is not meant to be a creepy time thing. You will sleep in your own room. It is a huge relief to have a balance to our fractured household. Please consider this.

[Redacted]
 [Redacted]
 [Redacted]
 [Redacted]

QUERY GRAPH MODEL



Get the suppliers and parts information for which the supplier's price is less than that of all other suppliers.

```

SELECT DISTINCT q1.partno, q1.descr, q2.suppno
FROM inventory AS q1, quotations AS q2
WHERE q1.partno = q2.partno
AND q1.descr = 'engine'
AND q1.price <= ALL(
SELECT q3.price
FROM quotations AS q3
WHERE q2.partno = q3.partno );

```



[DB Mistakes] Typo optimization)



From: [redacted]
 To: db-
 Sender: Db-
 List-Id: <db-
 Date: 1/2
 Spam Status: Sp

Hi,

It seems to me that in slide #

```

SELECT DISTINCT q1.partno
FROM inventory AS q1, quo
WHERE q1.partno = q2.part
AND q1.descr = 'engine'
AND *q1.price * <= ALL(
SELECT q3.price
FROM quotations AS q3
WHERE q2.partno = q3.pa

```

"*q1*.price <= ALL(..." sh
the quotation with the lo

Thank you for making th
absolute blast for me.

Best wishes,

Source: [Hamid Pirahesh](#)

LAST CLASS

IBM Starburst's stratified search approach.

- Query Rewriting (QGM to QGM)
- Plan Enumeration (QGM to STARs to LOLEPOPs)

One important aspect of Starburst is the introduction of properties as first-class components of a query plan and rules.

- **Relational:** Tables and columns accessed
- **Physical:** Tuple ordering, data location
- **Estimated:** Cardinalities, execution cost

OPTIMIZER GENERATORS

Framework to allow a DBMS implementer to write the rules for optimizing queries.

- Separate the search strategy from the data model.
- Separate the transformation rules and logical operators from physical rules and physical operators.

The implementation of the optimizer's pattern matching method and transformation rules can be independent of its search strategy.

TODAY'S AGENDA

EXODUS Optimizer Generator (1987)

Volcano Optimizer Generator (1994)

EXODUS OPTIMIZER

Optimizer generator framework for the Wisconsin EXODUS extensible DBMS project.

Rule-based approach that separates concerns between optimization logic and data structures.

- Translates algebraic transformation rules into executable optimizer code.
- Avoids exhaustive search by using dynamic programming and branch-and-bound pruning.
- Modifies search priorities based on past experience.
- **Bottom-up approach** (forward chaining).



Graefe



DeWitt



EXODUS: FIRST IMPLEMENTATION

Initially implemented rule-based optimizer using a logic programming language (Prolog, LOOP).

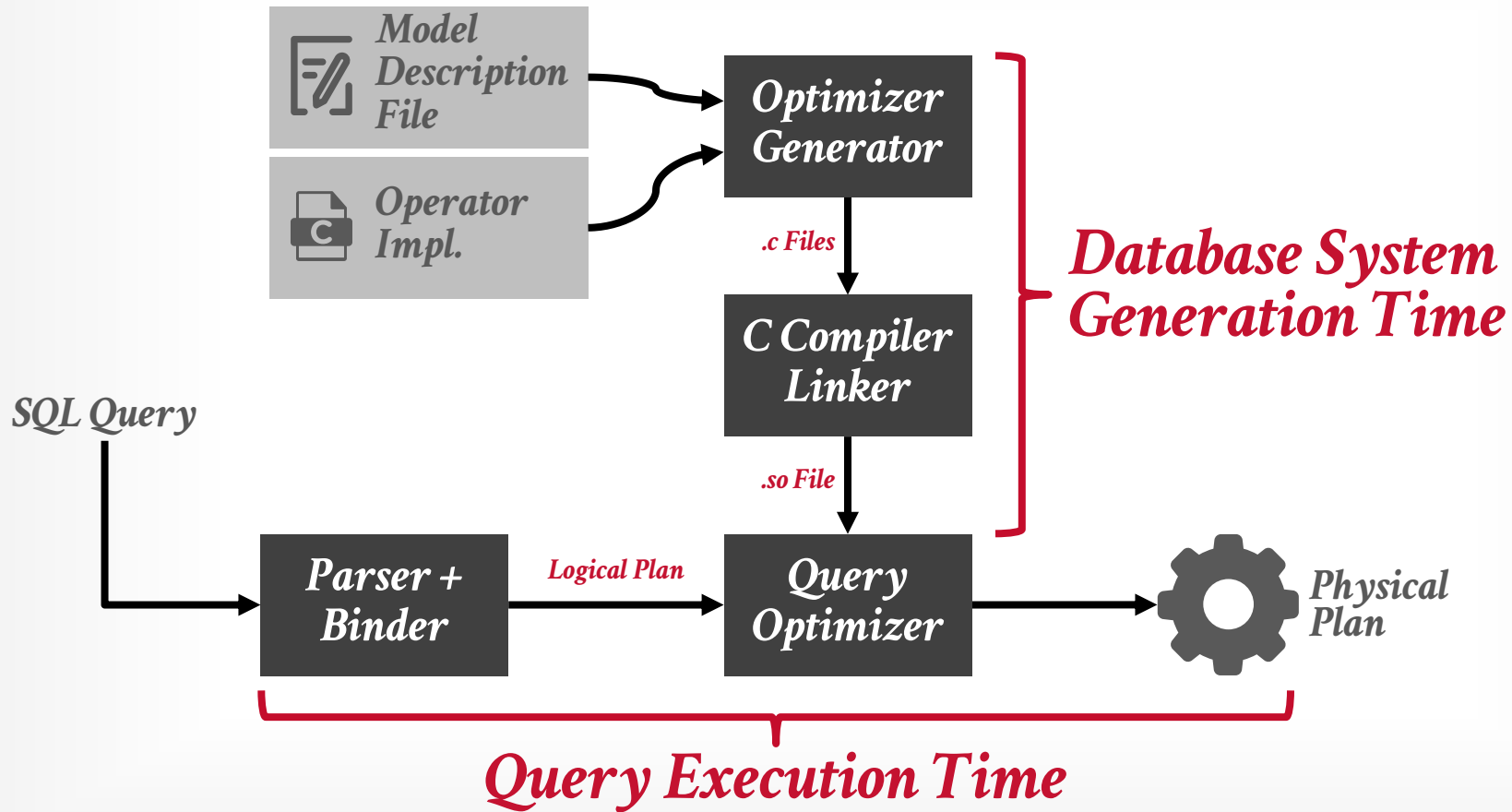
→ Such PLs have built-in support for pattern matching and include a search engine runtime.

Wisconsin DB team abandoned Prolog prototype:

→ Prolog's depth-first search was fixed and could not easily be change dynamically at runtime.

→ Their Prolog interpreter was too slow.

EXODUS: QUERY OPTIMIZER PIPELINE



EXODUS: MODEL DESCRIPTION FILE

Operator definitions.


Access method definitions.

Rules for transforming query plans.

Rules for correspondence between operators and methods.

EXODUS: RULES

Transformation Rules (Logical to Logical):

- Algebraic rules of expression equivalence (commutativity, associativity).
- Include hints to prevent reapplying the same rule to its output to avoid infinite loops.
- Example: JOIN(A, B)  JOIN(B, A)

Implementation Rules (Logical to Physical):

- Mapping of one or more logical operators to one or more physical operators.
- Supported bidirectional rules.
- Example: JOIN(A, B) ► HASH_JOIN(A, B)


EXODUS: SEARCH ALGORITHM

State Data Structures:

- **MESH**: In-memory hash table that holds remaining access plans for a query.
- **OPEN**: Priority queue of transformations to apply on MESH contents.

At the beginning of each round, the optimizer chooses the transformation rule it estimates will provide the largest cost improvement.

→ **Expected Cost Factor**



```
while (OPEN is not empty):  
    Select a transformation rule  
    from OPEN  
  
    Apply transformation rule to  
    the correct node(s) in MESH  
  
    Apply implementation rules &  
    cost analysis for new nodes  
  
    Add newly enabled  
    transformations to OPEN
```

EXODUS: EXPECTED COST FACTOR

If the query plan cost before a transformation is C and the expected cost factor of the transformation rule is f , then the query plan cost after applying the transformation is $C \times f$.

- Predicate pushdowns will be $f < 1$
- "Neutral" rules will be $f = 1$

The optimizer learns each rule's factor on its own from past experiences via propagation adjustments.

- Adjust the last two rules after an advantageous transformation.

EXODUS: EXPECTED COST FACTOR

If the query p and the expected cost rule is f , the transformation r

→ Predicate pushdown
→ "Neutral" rule

The optimization from past examples

→ Adjust the transformation

geometric sliding average $f \leftarrow (f^K * q)^{\frac{1}{K+1}}$	geometric mean $f \leftarrow (f^c * q)^{\frac{1}{c+1}}$
arithmetic sliding average $f \leftarrow \frac{f * K + q}{K+1}$	arithmetic mean $f \leftarrow \frac{f * c + q}{c+1}$

In these formulae, f is the expected cost factor for the rule under consideration, q is the current observed quotient of new cost over old cost, c is the count of how many times this rule has been applied so far, and K is the sliding average constant. As will be discussed below, all of these averaging formulas lead to statistically valid constructs, and the performance differences between them are fairly small.

OBSERVATION

EXODUS stores each logical operator with its physical operator(s) together in MESH.

→ Had to duplicate logical operators for each unique physical operator mapping.

Embed logic about adding additional operators to enforce properties in cost functions.

Upon invoking a logical transformation rule, the optimizer immediately applies implementation rules and performs cost analysis.

VOLCANO OPTIMIZER

General purpose cost-based query optimizer, based on equivalence rules on algebras.

- Easily add new operations and equivalence rules.
- Treats physical properties of data as first-class entities during planning.
- Same rule compilation pipeline as EXODUS.
- **Top-down approach** (backward chaining) using branch-and-bound search.



Graefe

Example: Academic prototypes, Calcite



THE VOLCANO OPTIMIZER GENERATOR:
EXTENSIBILITY AND EFFICIENT SEARCH
ICDE 1993

VOLCANO: OPTIMIZER DESIGN GOALS

Goal #1: Interoperable with existing DBMSs

Goal #2: Computational & storage efficient

Goal #3: Extensible physical properties

Goal #4: Extensible search guidance & pruning

Goal #5: Flexible cost model that supports incomplete queries.

VOLCANO: OVERVIEW

Uses two algebras (logical & physical) to map logical expressions to physical expressions.

Each rule is independent. Rely on search engine to find useful combinations of rules.

→ DBMS implementors should not manually combine rules.

Better to have fast search through compilation versus runtime augmentation via interpretation.

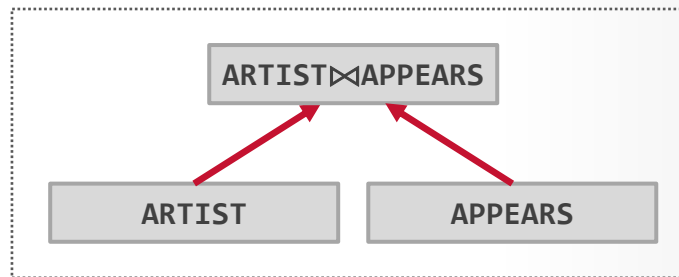
□ *Logical Op*

■ *Physical Op*

VOLCANO: EXPRESSIONS

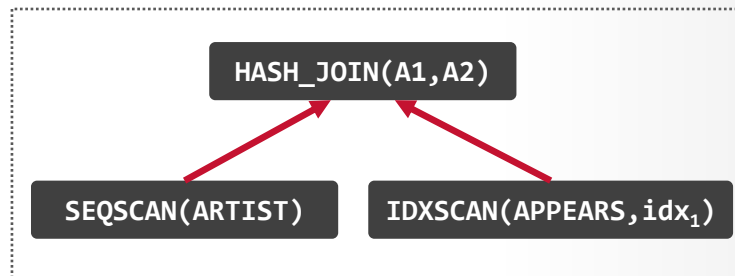
Logical Expressions:

→ Represent user queries as a directed tree of one or more logical operators (e.g., select, project, join).



Physical Expressions:

→ Represent query evaluation plans using physical algorithms (e.g., sort-merge join, hash join, nested loop join).



VOLCANO: OPERATOR DEFINITIONS

Logical Operators

- Properties: Schema, cardinality
- Property Function

Physical Operators:

- Properties: Sort-order, partitioning
- Cost + Property Functions
- Applicability Function: Whether it satisfies required properties of its logical operator(s)

Enforcer Operators:

- Property Function

VOLCANO: ENFORCER OPERATORS

The optimizer can inject "virtual" physical operators into a plan that make sure physical properties are satisfied.

→ Similar to Starburst's Glue operators.

DBMS implementor provides Volcano with a mapping of which properties an Enforcer operator will handle.

VOLCANO: RULES

Pattern matching function with an action function to permute a query plan. Similar to EXODUS rules.
→ Parameterized conditionals based on operator types.

Volcano rules also support auxiliary functions for conditional checks.

→ Procedural code that performs additional analysis of a query plan after a rule's pattern matches.

VOLCANO: COST FUNCTIONS

Represent cost as an abstract data type (ADT) that supports basic arithmetic and comparison functions.

DBMS implementor must provide the cost function calculation for each physical operator and enforcer.

→ Logical operators have no cost.

VOLCANO: SEARCH ENGINE

Dynamic programming based on backward chaining approach that supports general algebraic query and request optimization.

→ Top-down, goal-oriented control strategy.

Maintain a hash table ("lookup table") with expressions and equivalence classes.

→ Always check whether a logical or physical operator already exists in the lookup table.

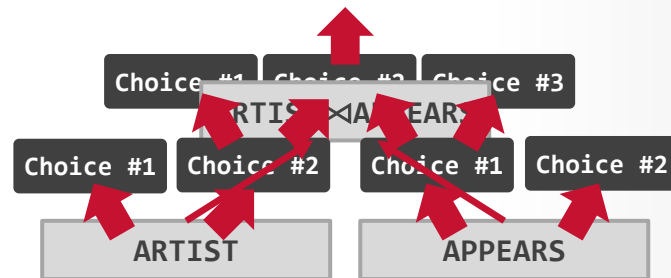
→ Allows optimizer to reuse cost calculations without needing to re-analyze sub-trees.

- Logical Op
- Physical Op

FORWARD VS. BACKWARD CHAINING

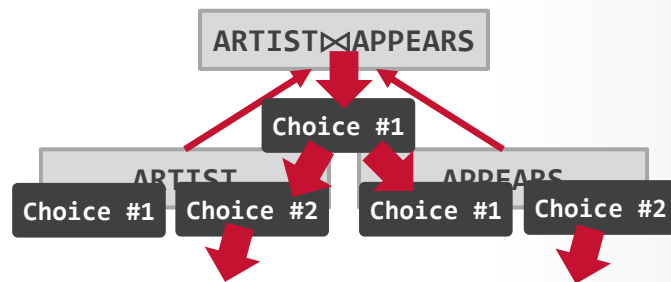
Forward Chaining:

- Start from query plan roots, trigger all rules that match those operators, and adds their conclusion to the known facts.
- Repeats until full query is generated.
- Breadth-first Search.



Backward Chaining:

- Start from the query result and works backward to determine what operators to add to the query plan to achieve result.
- Depth-first Search.



VOLCANO: SEARCH

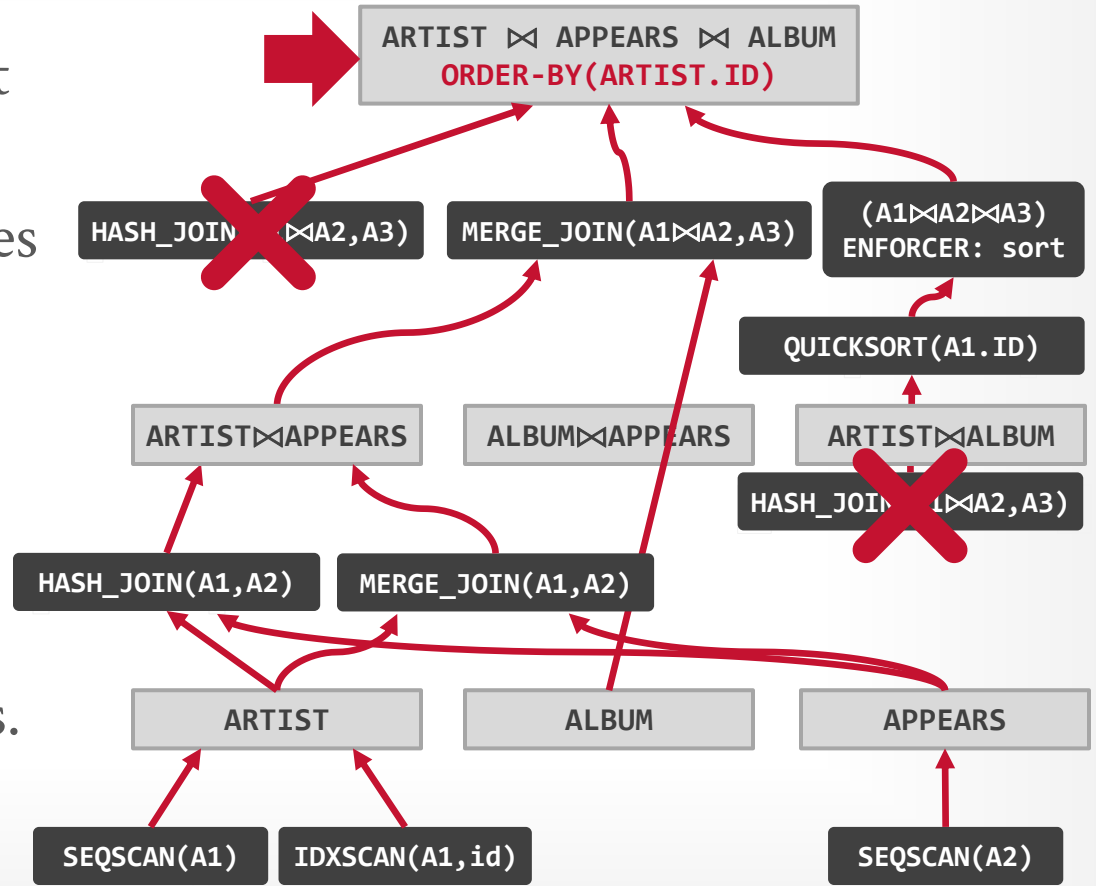
- Logical Op
- Physical Op

Start with a logical plan of what we want the query to be.

Invoke rules to create new nodes and traverse tree.

- Logical→Logical:
JOIN(A, B) to JOIN(B, A)
- Logical→Physical:
JOIN(A, B) to HASH_JOIN(A, B)

Enforcer rules require input to have certain properties.

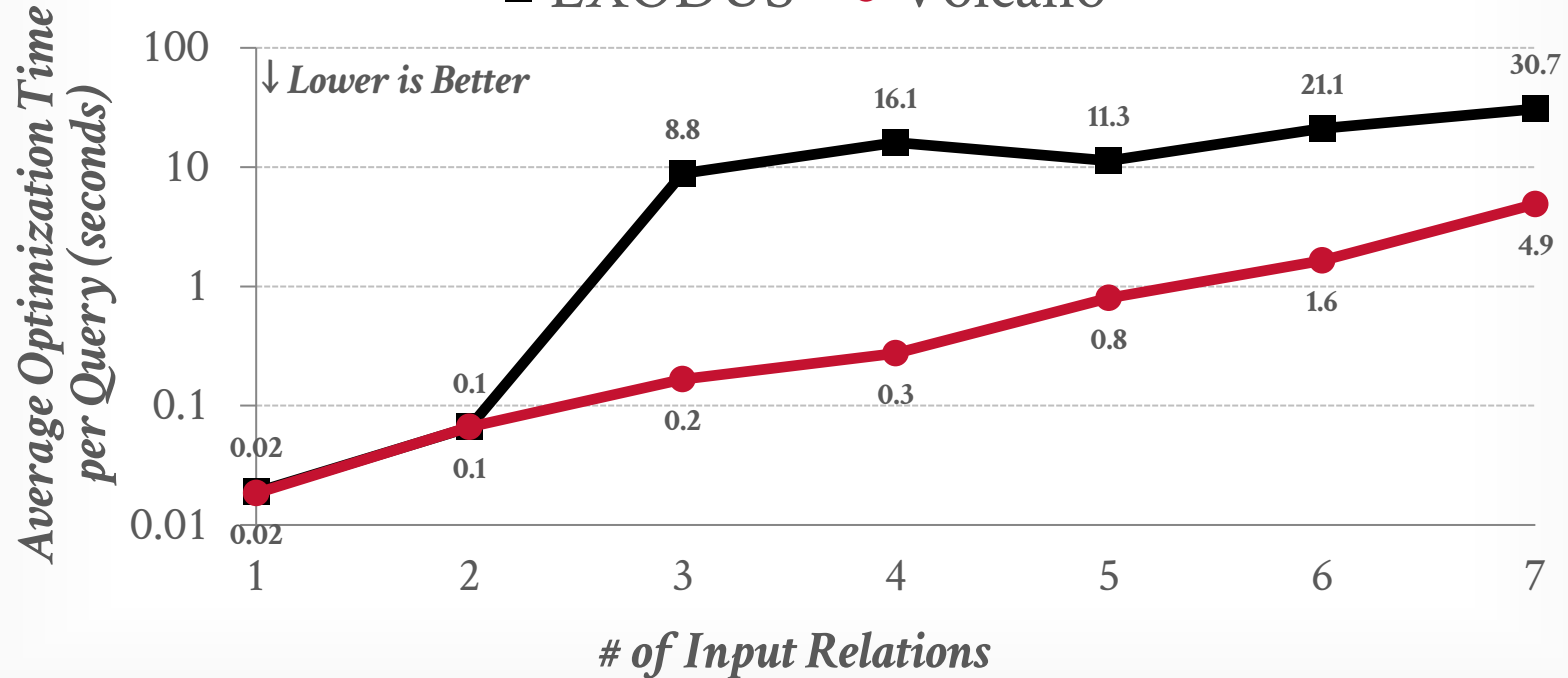


OPTIMIZER PERFORMANCE

Sun SPARCStation-1 (20mhz / 12 MIPS)

50 queries per complexity level

■ EXODUS ● Volcano



Source: [Goetz Graefe](#)

VOLCANO VS. STARBURST

Starburst grammar depends on hierarchy of intermediate levels.

→ More challenging to introduce new rules that fit into this complex hierarchy.

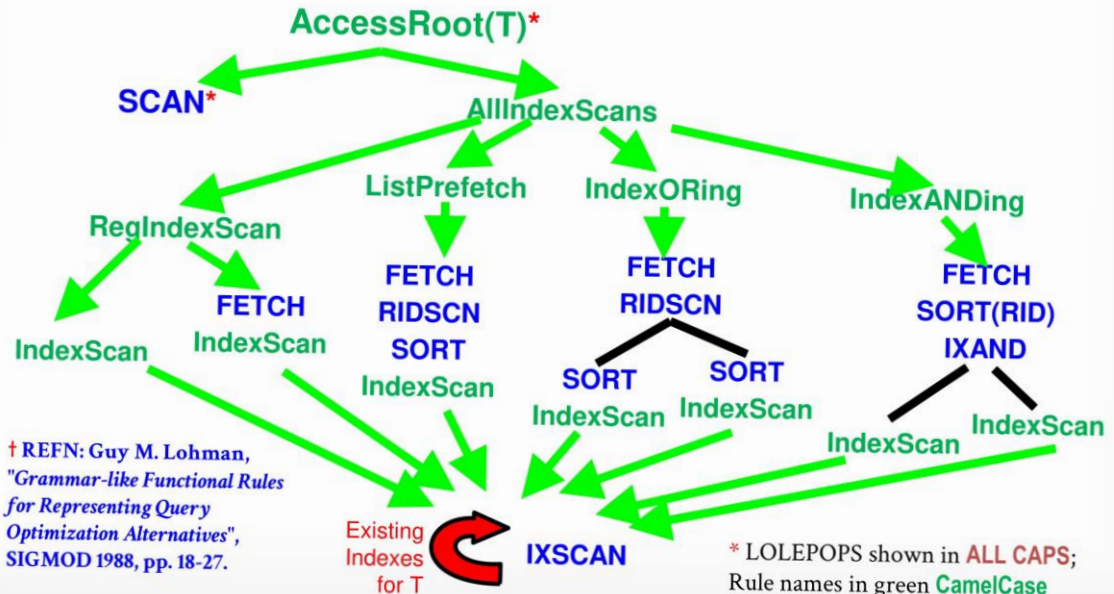
Volcano contends a single-level of rules reduces engineering complexity.

→ Can still do heuristic rewrites by give logical transformations a higher priority in the beginning of optimization process.

VOLCANO VS STARBUK

Generation of Table Access Alternatives

- Rules specify one or more alternatives, like a grammar[†]
- Each alternative specifies a nesting of other rules or LOLEPOPs*
- Can have iterators (e.g. all indexes for a table – see red arrow)



Source: [Guy Lohman](#)

SPECIAL TOPICS (SPRING 2025)

VOLCANO OPTIMIZER

Advantages:

- Compile declarative rules to generate transformations.
- Better extensibility with an efficient search engine. Reduce redundant estimations using memoization.

Disadvantages:

- All equivalence classes are completely expanded to generate all possible logical operators before the optimization search.
- Not easy to modify predicates.

PARTING THOUGHTS

Key problem with Volcano is it expands equivalence classes to generate all possible transformations before the traversing below the search tree.

Volcano paper mentions that it may be possible to reuse optimization results across multiple queries.

PARTING THOUGHTS

Key problem
equivalence
transformat
search tree.

Volcano pa
reuse optim

rithm uses backward chaining, because it explores only those subqueries and plans that truly participate in a larger expression. We call our search algorithms *directed dynamic programming*.

Dynamic programming is used in optimizers created with the Volcano optimizer generator by retaining a large set of partial optimization results and using these earlier results in later optimization decisions. Currently, this set

of partial optimization results is reinitialized for each query being optimized. In other words, earlier partial optimization results are used during the optimization of only a single query. We are considering research into longer-lived partial results in the future.

Algebraic transformation systems always include the possibility of deriving the same expression in several different ways. In order to prevent redundant optimization effort by detecting redundant (i.e., multiple equivalent)

NEXT CLASS

Cascades!!!

→ Read Chapter 2 but do not submit a reading synopsis.