Optimizer Implementation (Part II)

@Andy_Pavlo // 15-721 // Spring 2019
DATABASE TALKS

Amazon Redshift
→ Ippokratis Pandis (PhD'07)
→ Today @ 4:30pm
→ DH 2315

SAP HANA
→ Anil Goel
→ Thursday May 2\textsuperscript{nd} @ 12:00pm
→ CIC - 4th floor (ISTC Panther Hollow Room)
TODAY’S AGENDA

Cascades / Columbia
Plan Enumeration
Other Implementations
QUERY OPTIMIZATION STRATEGIES

Choice #1: Heuristics
→ INGRES, Oracle (until mid 1990s)

Choice #2: Heuristics + Cost-based Join Search
→ System R, early IBM DB2, most open-source DBMSs

Choice #3: Randomized Search
→ Academics in the 1980s, current Postgres

Choice #4: Stratified Search
→ IBM’s STARBURST (late 1980s), now IBM DB2 + Oracle

Choice #5: Unified Search
→ Volcano/Cascades in 1990s, now MSSQL + Greenplum
OPTIMIZER GENERATORS

Framework to allow a DBMS implementer to write the declarative 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.

Implementation can be independent of the optimizer's search strategy.

Examples: Starburst, Exodus, Volcano, Cascades, OPT++
STRATIFIED SEARCH

First rewrite the logical query plan using transformation rules.
→ The engine checks whether the transformation is allowed before it can be applied.
→ Cost is never considered in this step.

Then perform a cost-based search to map the logical plan to a physical plan.
UNIFIED SEARCH

Unify the notion of both logical → logical and logical → physical transformations.
→ No need for separate stages because everything is transformations.

This approach generates a lot more transformations so it makes heavy use of memoization to reduce redundant work.
TOP-DOWN VS. BOTTOM-UP

Top-down Optimization
→ Start with the final outcome that you want, and then work down the tree to find the optimal plan that gets you to that goal.
→ Example: Volcano, Cascades

Bottom-up Optimization
→ Start with nothing and then build up the plan to get to the final outcome that you want.
→ Examples: System R, Starburst
CASCADES OPTIMIZER

Object-oriented implementation of the Volcano query optimizer.

Simplistic expression re-writing can be through a direct mapping function rather than an exhaustive search.
CASCADES OPTIMIZER

Optimization tasks as data structures.
Rules to place property enforcers.
Ordering of moves by promise.
Predicates as logical/physical operators.
CASCades – EXPressions

A **expression** is an operator with zero or more input expressions.

```sql
SELECT * FROM A
JOIN B ON A.id = B.id
JOIN C ON C.id = A.id;
```

Logical Expression: \((A \bowtie B) \bowtie C\)

Physical Expression: \((A_{Seq} \bowtie_{HJ} B_{Seq}) \bowtie_{NL} C_{Seq}\)
CASCADES – GROUPS

A **group** is a set of logically equivalent logical and physical expressions that produce the same output.
→ All logical forms of an expression.
→ All physical expressions that can be derived from selecting the allowable physical operators for the corresponding logical forms.

<table>
<thead>
<tr>
<th>Output: [ABC]</th>
<th>Logical Exps</th>
<th>Physical Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. (A⨝B)⨝C</td>
<td>1. (A\text{Seq}_\text{NL}B\text{Seq})⨝\text{NL}C\text{Seq}</td>
</tr>
<tr>
<td></td>
<td>2. (B⨝C)⨝A</td>
<td>2. (B\text{Seq}_\text{NL}C\text{Seq})⨝\text{NL}A\text{Seq}</td>
</tr>
<tr>
<td></td>
<td>3. (A⨝C)⨝B</td>
<td>3. (A\text{Seq}_\text{NL}C\text{Seq})⨝\text{NL}B\text{Seq}</td>
</tr>
<tr>
<td></td>
<td>4. A⨝(B⨝C)</td>
<td>4. A\text{Seq}<em>\text{NL}(C\text{Seq}</em>\text{NL}B\text{Seq})</td>
</tr>
<tr>
<td></td>
<td>⋮</td>
<td>⋮</td>
</tr>
</tbody>
</table>
CASCADERS – GROUPS

A **group** is a set of logically equivalent logical and physical expressions that produce the same output.

→ All logical forms of an expression.
→ All physical expressions that can be derived from selecting the allowable physical operators for the corresponding logical forms.

<table>
<thead>
<tr>
<th>Logical Exps</th>
<th>Physical Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ((A \bowtie B) \bowtie C)</td>
<td>1. ((A_{seq} \bowtie_{NL} B_{seq}) \bowtie_{NL} C_{seq})</td>
</tr>
<tr>
<td>2. ((B \bowtie C) \bowtie A)</td>
<td>2. ((B_{seq} \bowtie_{NL} C_{seq}) \bowtie_{NL} A_{seq})</td>
</tr>
<tr>
<td>3. ((A \bowtie C) \bowtie B)</td>
<td>3. ((A_{seq} \bowtie_{NL} C_{seq}) \bowtie_{NL} B_{seq})</td>
</tr>
<tr>
<td>4. (A \bowtie (B \bowtie C))</td>
<td>4. (A_{seq} \bowtie_{NL} (C_{seq} \bowtie_{NL} B_{seq}))</td>
</tr>
</tbody>
</table>
A **group** is a set of logically equivalent logical and physical expressions that produce the same output.

→ All logical forms of an expression.

→ All physical expressions that can be derived from selecting the allowable physical operators for the corresponding logical forms.

<table>
<thead>
<tr>
<th>Logical Exps</th>
<th>Physical Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td>(A⨝B)⨝C</td>
<td>(A(\text{Seq}∸\text{NL})B(\text{Seq}∸\text{NL})C(\text{Seq}∸\text{NL}))</td>
</tr>
<tr>
<td>(B⨝C)⨝A</td>
<td>(B(\text{Seq}∸\text{NL})C(\text{Seq}∸\text{NL})A(\text{Seq}∸\text{NL}))</td>
</tr>
<tr>
<td>(A⨝C)⨝B</td>
<td>(A(\text{Seq}∸\text{NL})C(\text{Seq}∸\text{NL})B(\text{Seq}∸\text{NL}))</td>
</tr>
<tr>
<td>A⨝(B⨝C)</td>
<td>(\vdots)</td>
</tr>
</tbody>
</table>
CASCADES – MULTI-EXPRESSION

Instead of explicitly instantiating all possible expressions in a group, the optimizer implicitly represents redundant expressions in a group as a multi-expression.

→ This reduces the number of transformations, storage overhead, and repeated cost estimations.

<table>
<thead>
<tr>
<th>Output: [ABC]</th>
<th>Logical Multi-Exps</th>
<th>Physical Multi-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. [AB]⨝[C]</td>
<td>1. [AB]⨝_SM[C]</td>
</tr>
<tr>
<td></td>
<td>2. [BC]⨝[A]</td>
<td>2. [AB]⨝_HJ[C]</td>
</tr>
<tr>
<td></td>
<td>3. [AC]⨝[B]</td>
<td>3. [AB]⨝_NL[C]</td>
</tr>
<tr>
<td></td>
<td>⋮</td>
<td>⋮</td>
</tr>
</tbody>
</table>
CASCADeS – RULES

A **rule** is a transformation of an expression to a logically equivalent expression.

→ **Transformation Rule**: Logical to Logical
→ **Implementation Rule**: Logical to Physical

Each rule is represented as a pair of attributes:

→ **Pattern**: Defines the structure of the logical expression that can be applied to the rule.
→ **Substitute**: Defines the structure of the result after applying the rule.
CASCADES – RULES

Pattern

```
GROUP 1
  └── EQJOIN
    └── GROUP 2
```

- **Group**
- **Logical Expr**
- **Physical Expr**
CASCADES – RULES

Pattern

EQJOIN

EQJOIN

GROUP 3

GROUP 1

GROUP 2

[AB]⨝ C

A⨝B

GET(A)

GET(B)

GET(C)

Matching Plan
Pattern

CASCADAS – RULES

Transformation Rule
Rotate Left-to-Right

Matching Plan
CASCADeS — RULES

Pattern

Transformation Rule
Rotate Left-to-Right

Matching Plan

Implementation Rule
EQJOIN → SORTMERGE
CASCADeS – MEMO TABLE

Stores all previously explored alternatives in a compact graph structure / hash table.

Equivalent operator trees and their corresponding plans are stored together in groups.

Provides memoization, duplicate detection, and property + cost management.
PRINCIPLE OF OPTIMALITY

Every sub-plan of an optimal plan is itself optimal.

This allows the optimizer to restrict the search space to a smaller set of expressions.  
→ The optimizer never has to consider a plan containing sub-plan P1 that has a greater cost than equivalent plan P2 with the same physical properties.
# CASCADES – MEMO TABLE

<table>
<thead>
<tr>
<th>Best Expr</th>
<th>Output:</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ABC]</td>
<td>[ABC]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[AB]</td>
<td>[AB]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[A]</td>
<td>[A]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[B]</td>
<td>[B]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[C]</td>
<td>[C]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Output: Logical M-Exps Physical M-Exps

- **Output: [AB]**
  - Logical M-Exps
  - Physical M-Exps

- **Output: [BC]**
  - Logical M-Exps
  - Physical M-Exps

- **Output: [AC]**
  - Logical M-Exps
  - Physical M-Exps

- **Output: [B]**
  - Logical M-Exps
  - Physical M-Exps

Best Expr

Output: Logical M-Exps Physical M-Exps

- **Output: [A]**
  - 1. GET(A)
  - Logical M-Exps
  - Physical M-Exps

- **Output: [B]**
  - 1. GET(B)
  - Logical M-Exps
  - Physical M-Exps

- **Output: [C]**
  - 1. GET(C)
  - Logical M-Exps
  - Physical M-Exps

Output:

- [A]
- [B]
- [C]
CASCADES – MEMO TABLE

<table>
<thead>
<tr>
<th>Best Expr</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ABC]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[AB]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[A]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[B]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[C]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Output:** [ABC]
  - Logical M-Exps: 1. [AB]⨝[C]
  - Physical M-Exps: 2.

- **Output:** [AB]
  - Logical M-Exps: 1. [A]⨝[B]
  - Physical M-Exps: 2.

- **Output:** [A]
  - Logical M-Exps: 1. GET(A)
  - Physical M-Exps: 2.

- **Output:** [B]
  - Logical M-Exps: 1. GET(B)
  - Physical M-Exps: 2.

- **Output:** [C]
  - Logical M-Exps: 1. GET(C)
  - Physical M-Exps: 2.
**CASCADES – MEMO TABLE**

<table>
<thead>
<tr>
<th>Best Expr</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ABC]</td>
<td>Output: [ABC]</td>
<td>[AB] ⨝ [C]</td>
</tr>
<tr>
<td>[AB]</td>
<td>Logical M-Exps</td>
<td>Physical M-Exps</td>
</tr>
<tr>
<td>[A]</td>
<td>1. [A] ⨝ [B]</td>
<td></td>
</tr>
<tr>
<td>[B]</td>
<td>[A] ⨝ [B]</td>
<td></td>
</tr>
<tr>
<td>[C]</td>
<td>Output: [C]</td>
<td>Logical M-Exps</td>
</tr>
<tr>
<td></td>
<td>1. GET(C)</td>
<td>Physical M-Exps</td>
</tr>
</tbody>
</table>

Output: [A] 1. GET(A)  
Physical M-Exps

Output: [B] 1. GET(B)  
Physical M-Exps

Best Expr

[ABC]  
[AB]  
[A]  
[B]  
[C]  

Logical M-Exps  
Physical M-Exps

Output: [ABC]  
Logical M-Exps  
Physical M-Exps

Output: [AB]  
Logical M-Exps  
Physical M-Exps

Output: [C]  
Logical M-Exps  
Physical M-Exps

Output: [A]  
Logical M-Exps  
Physical M-Exps

Output: [B]  
Logical M-Exps  
Physical M-Exps
CASCADES – MEMO TABLE

<table>
<thead>
<tr>
<th>Best Expr</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ABC]</td>
<td>Output: [ABC]</td>
<td>Logical M-Exps</td>
</tr>
<tr>
<td></td>
<td>1. [AB]⨝[C]</td>
<td>Physical M-Exps</td>
</tr>
<tr>
<td>[AB]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[A]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[B]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[C]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Output: [ABC]

Logical M-Exps
1. [AB]⨝[C]

Physical M-Exps

Output: [AB]

Logical M-Exps
1. [A]⨝[B]

Physical M-Exps

Output: [A]

Logical M-Exps
1. GET(A)

Physical M-Exps
1. SeqScan(A)
2. IdxScan(A)

Output: [B]

Logical M-Exps
1. GET(B)

Physical M-Exps

Output: [C]

Logical M-Exps
1. GET(C)

Physical M-Exps

Best Expr

[ABC]

[AB]

[A]

[B]

[C]
CASCADES – MEMO TABLE

<table>
<thead>
<tr>
<th>Best Exp</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ABC]</td>
<td>[AB] △ [C]</td>
<td></td>
</tr>
<tr>
<td>[AB]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[A]</td>
<td>SeqScan(A)</td>
<td></td>
</tr>
<tr>
<td>[B]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[C]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cost: 10

Output: [A]
1. GET(A)
2. SeqScan(A)

Output: [B]
1. GET(B)
2. IdxScan(A)

Output: [C]
1. GET(C)
2. SeqScan(C)

Output: [AB]
1. [AB] △ [B]

Output: [ABC]
1. [ABC] △ [C]

Logical M-Exps
Physical M-Exps
CASCADES – MEMO TABLE

Best Exp

<table>
<thead>
<tr>
<th>[ABC]</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[ABC]</td>
<td></td>
</tr>
<tr>
<td>[AB]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[A]</td>
<td>SeqScan(A)</td>
<td></td>
</tr>
<tr>
<td>[B]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[C]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Output: [ABC]

Logical M-Exps: 1. [AB]⨝ [C]

Physical M-Exps: 

Output: [AB]

Logical M-Exps: 1. [A]⨝ [B]

Physical M-Exps: 

Output: [A]

Logical M-Exps: 1. GET(A)

Physical M-Exps: 

Output: [B]

Logical M-Exps: 1. GET(B)

Physical M-Exps: 1. SeqScan(B) 2. IdxScan(B)

Output: [C]

Logical M-Exps: 1. GET(C)

Physical M-Exps: 

Cost: 10

Output: [A]

Logical M-Exps: 1. GET(A)

Physical M-Exps: 1. SeqScan(A) 2. IdxScan(A)

Output: [B]

Logical M-Exps: 1. GET(B)

Physical M-Exps: 1. SeqScan(B) 2. IdxScan(B)
### CASCADES – MEMO TABLE

<table>
<thead>
<tr>
<th>Best Expr</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ABC]</td>
<td>Output: [ABC]</td>
<td></td>
</tr>
<tr>
<td>[AB]</td>
<td>Logical M-Exps 1. [AB]⨝[C]</td>
<td>Physical M-Exps</td>
</tr>
<tr>
<td>[A]</td>
<td>SeqScan(A)</td>
<td></td>
</tr>
<tr>
<td>[B]</td>
<td>SeqScan(B)</td>
<td></td>
</tr>
<tr>
<td>[C]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Output:**

1. [AB] ⨝ [C]

**Cost:** 10

<table>
<thead>
<tr>
<th>Output: [A]</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. GET(A)</td>
<td>Logical M-Exps 1. SeqScan(A)</td>
<td>Physical M-Exps 1. SeqScan(A)</td>
</tr>
<tr>
<td></td>
<td>2. IdxScan(A)</td>
<td></td>
</tr>
</tbody>
</table>

**Cost:** 20

<table>
<thead>
<tr>
<th>Output: [B]</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. GET(B)</td>
<td>Logical M-Exps 1. SeqScan(B)</td>
<td>Physical M-Exps 1. SeqScan(B)</td>
</tr>
<tr>
<td></td>
<td>2. IdxScan(B)</td>
<td></td>
</tr>
</tbody>
</table>
## CASCADES – MEMO TABLE

<table>
<thead>
<tr>
<th>Best Expr</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ABC]</td>
<td>[ABC]</td>
<td></td>
</tr>
<tr>
<td>[AB]</td>
<td>[AB] ⨝ [C]</td>
<td></td>
</tr>
<tr>
<td>[A]</td>
<td>[A] ⨝ [B]</td>
<td>NL C</td>
</tr>
<tr>
<td>[B]</td>
<td>[B] ⨝ [A]</td>
<td>NL A</td>
</tr>
<tr>
<td>[C]</td>
<td>[C] ⨝ [B]</td>
<td>NL B</td>
</tr>
</tbody>
</table>

Cost: 10

<table>
<thead>
<tr>
<th>Output: [A]</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GET(A)</td>
<td>SeqScan(A)</td>
</tr>
<tr>
<td></td>
<td>IdxScan(A)</td>
<td></td>
</tr>
</tbody>
</table>

Cost: 20

<table>
<thead>
<tr>
<th>Output: [B]</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GET(B)</td>
<td>SeqScan(B)</td>
</tr>
<tr>
<td></td>
<td>IdxScan(B)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output: [C]</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GET(C)</td>
<td></td>
</tr>
</tbody>
</table>

Cost: 20
### CASCADES – MEMO TABLE

<table>
<thead>
<tr>
<th>Best Expr</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ABC]</td>
<td>[ABC]</td>
<td></td>
</tr>
<tr>
<td>[AB]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[A]</td>
<td>SeqScan(A)</td>
<td></td>
</tr>
<tr>
<td>[B]</td>
<td>SeqScan(B)</td>
<td></td>
</tr>
<tr>
<td>[C]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Output: [AB]

**Logical M-Exps**
1. [A]⨝ [B]
2. [B]⨝ [A]

**Physical M-Exps**
1. [A]⨝ \text{NL}[B]
2. [A]⨝ \text{SM}[B]
3. [B]⨝ \text{NL}[A]

- **Cost: 10**

Output: [A]

Logical M-Exps
1. GET(A)

Physical M-Exps
1. SeqScan(A)
2. IdxScan(A)

Output: [B]

Logical M-Exps
1. GET(B)

Physical M-Exps
1. SeqScan(B)
2. IdxScan(B)

- **Cost: 20**

Output: [C]

Logical M-Exps
1. GET(C)

Physical M-Exps
1. 

- **Cost: 20**

**Best Expr**

- [ABC]
- [AB]
- [A] SeqScan(A)
- [B] SeqScan(B)
- [C]

**Costs**

- 10
- 20

**Best Expr**

- SeqScan(A)
- SeqScan(B)
## CASCADES – MEMO TABLE

<table>
<thead>
<tr>
<th>Best Expr</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ABC]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[AB]</td>
<td>[A]⨝_{SM}[B]</td>
<td></td>
</tr>
<tr>
<td>[A]</td>
<td>SeqScan(A)</td>
<td></td>
</tr>
<tr>
<td>[B]</td>
<td>SeqScan(B)</td>
<td></td>
</tr>
<tr>
<td>[C]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Output: [ABC]

**Logical M-Exps**
1. [AB]⨝[C]

**Physical M-Exps**

Output: [AB]

**Logical M-Exps**
1. [A]⨝[B]
2. [B]⨝[A]

**Physical M-Exps**
1. [A]⨝_{NL}[B]
2. [A]⨝_{SM}[B]
3. [B]⨝_{NL}[A]

**Cost:** 50+(10+20)

Output: [A]

**Logical M-Exps**
1. GET(A)

**Physical M-Exps**
1. SeqScan(A)
2. IdxScan(A)

**Cost:** 10

Output: [B]

**Logical M-Exps**
1. GET(B)

**Physical M-Exps**
1. SeqScan(B)
2. IdxScan(B)

**Cost:** 20
## CASCADES – MEMO TABLE

<table>
<thead>
<tr>
<th>Best Expr</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ABC]</td>
<td>Output: [ABC]</td>
<td>Logical M-Exps 1. [AB]⨝[C]</td>
</tr>
<tr>
<td>[A]</td>
<td>2. [B]⨝[A]</td>
<td>2. [A]⨝_{SM}[B]</td>
</tr>
<tr>
<td>[B]</td>
<td></td>
<td>3. [B]⨝_{NL}[A]</td>
</tr>
<tr>
<td>[C]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Output: [AB]

Cost: 50+(10+20)

Cost: 5

Cost: 10

Output: [A]

Logical M-Exps 1. GET(A)

Physical M-Exps 1. SeqScan(A)

idxScan(A)

Cost: 20

Output: [B]

Logical M-Exps 1. GET(B)

Physical M-Exps 1. SeqScan(B)

idxScan(B)

Cost: 10

Output: [A]

Logical M-Exps 1. GET(A)

Physical M-Exps 1. SeqScan(A)

idxScan(A)

Cost: 20

Output: [B]

Logical M-Exps 1. GET(B)

Physical M-Exps 1. SeqScan(B)

idxScan(B)
### CASCADES – MEMO TABLE

<table>
<thead>
<tr>
<th>Best Expr</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ABC]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[AB]</td>
<td>[A]⨝_{SM}[B]</td>
<td></td>
</tr>
<tr>
<td>[A]</td>
<td>SeqScan(A)</td>
<td></td>
</tr>
<tr>
<td>[B]</td>
<td>SeqScan(B)</td>
<td></td>
</tr>
<tr>
<td>[C]</td>
<td>IdxScan(C)</td>
<td></td>
</tr>
</tbody>
</table>

#### Cost: 50+(10+20)

<table>
<thead>
<tr>
<th>Output: [ABC]</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. [AB]⨝[C]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. [BC]⨝[A]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. [AC]⨝[B]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4. [B]⨝[AC]</td>
<td></td>
</tr>
</tbody>
</table>

#### Cost: 5 |

<table>
<thead>
<tr>
<th>Output: [C]</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. GET(C)</td>
<td></td>
</tr>
</tbody>
</table>

#### Cost: 10 |

<table>
<thead>
<tr>
<th>Output: [A]</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. GET(A)</td>
<td>SeqScan(A)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IdxScan(A)</td>
</tr>
</tbody>
</table>

#### Cost: 20 |

<table>
<thead>
<tr>
<th>Output: [B]</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. GET(B)</td>
<td>SeqScan(B)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IdxScan(B)</td>
</tr>
</tbody>
</table>
### CASCADeS – MEMO TABLE

<table>
<thead>
<tr>
<th>Best Expr</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ABC]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[AB]</td>
<td>[A]⨝_{SM}[B]</td>
<td></td>
</tr>
<tr>
<td>[A]</td>
<td>SeqScan(A)</td>
<td></td>
</tr>
<tr>
<td>[B]</td>
<td>SeqScan(B)</td>
<td></td>
</tr>
<tr>
<td>[C]</td>
<td>IdxScan(C)</td>
<td></td>
</tr>
</tbody>
</table>

**Cost: 50+(10+20)**

<table>
<thead>
<tr>
<th>Output: [ABC]</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. [AB]⨝[C]</td>
<td>1. [AB]⨝_{NL}C</td>
</tr>
<tr>
<td></td>
<td>2. [BC]⨝[A]</td>
<td>2. [BC]⨝_{NL}A</td>
</tr>
<tr>
<td></td>
<td>3. [AC]⨝[B]</td>
<td>3. [AC]⨝_{NL}B</td>
</tr>
<tr>
<td></td>
<td>4. [B]⨝[AC]</td>
<td></td>
</tr>
</tbody>
</table>

**Cost: 5**

<table>
<thead>
<tr>
<th>Output: [AB]</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. GET(A)</td>
<td>1. SeqScan(C)</td>
</tr>
<tr>
<td></td>
<td>2. IdxScan(C)</td>
<td>2. IdxScan(C)</td>
</tr>
</tbody>
</table>

**Cost: 10**

<table>
<thead>
<tr>
<th>Output: [AB]</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. GET(A)</td>
<td>SeqScan(C)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output: [BC]</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. GET(C)</td>
<td>SeqScan(C)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output: [AC]</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. SeqScan(A)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. IdxScan(B)</td>
<td>IdxScan(B)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output: [AC]</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. [A]⨝[C]</td>
<td>SeqScan(B)</td>
</tr>
<tr>
<td></td>
<td>2. [C]⨝[A]</td>
<td>IdxScan(B)</td>
</tr>
</tbody>
</table>

**Cost: 5**

<table>
<thead>
<tr>
<th>Output: [C]</th>
<th>Logical M-Exps</th>
<th>Physical M-Exps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. GET(C)</td>
<td>SeqScan(C)</td>
</tr>
<tr>
<td></td>
<td>2. IdxScan(C)</td>
<td>IdxScan(C)</td>
</tr>
</tbody>
</table>
SEARCH TERMINATION

Approach #1: Wall-clock Time
→ Stop after the optimizer runs for some length of time.

Approach #2: Cost Threshold
→ Stop when the optimizer finds a plan that has a lower cost than some threshold.

Approach #3: Transformation Exhaustion
→ Stop when there are no more ways to transform the target plan. Usually done per group.
CASCADES IMPLEMENTATIONS

Standalone:
→ **Wisconsin OPT++** (1990s)
→ **Portland State Columbia** (1990s)
→ **Pivotal Orca** (2010s)
→ **Apache Calcite** (2010s)

Integrated:
→ **Microsoft SQL Server** (1990s)
→ **Tandem NonStop SQL** (1990s)
→ **Clustrix** (2000s)
→ **CMU Peloton** (2010s)
OBSERVATION

All of the queries we have looked at so far have had the following properties:
→ Equi/Inner Joins
→ Simple join predicates that reference only two tables.
→ No cross products

Real-world queries are much more complex:
→ Outer Joins
→ Semi-joins
→ Anti-joins
REORDERING LIMITATIONS

No valid reordering is possible.

SELECT * FROM A
LEFT OUTER JOIN B
ON A.id = B.id
FULL OUTER JOIN C
ON B.val = C.id);

Source: Pit Fender
No valid reordering is possible.

The $\text{A} \bowtie \text{B}$ operator is not commutative with $\text{B} \bowtie \text{C}$.

$\rightarrow$ The DBMS does not know the value of $\text{B.val}$ until after computing the join with $\text{A}$.

Source: Pit Fender
How to generate different join orderings to feed into the optimizer's search model.
→ Need to be efficient to not slowdown the search.

Approach #1: Generate-and-Test
Approach #2: Graph Partitioning
Dynamic Programming Strikes Back

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

ABSTRACT

Two optimal algorithms for dynamic programming are known for solving certain combinatorial optimization problems. The first, which is based on the concept of a primitive, is a bottom-up, tree-based, dynamic programming algorithm. The second, which is based on the concept of a bipartite graph, is a top-down, graph-based, dynamic programming algorithm. For each, we give a worst-case complexity analysis of the algorithm. The algorithms are applied to the problem of finding the shortest path in a graph. We also show that the first algorithm is a special case of the second algorithm.

Category and Subject Descriptors

B.4.3 [Proceedings]: Dynamic Programming

General Terms

Algorithms, Theory

1. INTRODUCTION

For the past several years, the performance of a database management system, the cost-benefit gap appears to be an essential issue. For the past several years, the performance of a database management system, general structure, the cost-benefit gap appears to be an essential issue. General structure, the cost-benefit gap appears to be an essential issue.

2. PRELIMINARIES

Before we give the top-down algorithm, we must consider some basic properties of dynamic programming. A dynamic programming algorithm is a special case of a top-down algorithm. A top-down algorithm is a special case of a bottom-up algorithm.

SIGMOD 2008

Counter Strike: Generic Top-Down Join Enumeration for Hypergraphs

Piotr Indyk
University of California
Los Angeles

ABSTRACT

Finding the optimal execution plan for a join operation in a database is a fundamental challenge. We present a new approach to finding the optimal execution plan for a join operation in a database. Our approach is based on a top-down method. Our algorithm is based on a top-down method.

Category and Subject Descriptors

F.2.1 [Proceedings]: Data Structures

2. PRELIMINARIES

Before we give the top-down algorithm, we must consider some basic properties of dynamic programming. A dynamic programming algorithm is a special case of a top-down algorithm. A top-down algorithm is a special case of a bottom-up algorithm.

VLDB 2013

GERMANS
DYNAMIC PROGRAMMING OPTIMIZER

Model the query as a hypergraph and then incrementally expand to enumerate new plans.

Algorithm Overview:
→ Iterate connected sub-graphs and incrementally add new edges to other nodes to complete query plan.
→ Use rules to determine which nodes the traversal is allowed to visit and expand.
Predicates are defined as part of each operator. 
→ These are typically represented as an AST.
→ Postgres implements them as flatten lists.

The same logical operator can be represented in multiple physical operators using variations of the same expression.
PREDICATE PUSHDOWN

Approach #1: Logical Transformation
→ Like any other transformation rule in Cascades.
→ Can use cost-model to determine benefit.

Approach #2: Rewrite Phase
→ Perform pushdown before starting search using an initial rewrite phase. Tricky to support complex predicates.

Approach #3: Late Binding
→ Perform pushdown after generating optimal plan in Cascades. Will likely produce a bad plan.
Predicate Migration

Observation: Not all predicates cost the same to evaluate on tuples.

```
SELECT * FROM foo
WHERE foo.id = 1234
AND SHA_512(foo.val) = '....'
```

The optimizer should consider selectivity and computation cost when determining the evaluation order of predicates.
PIVOTAL ORCA

Standalone Cascades implementation.
→ Originally written for Greenplum.
→ Extended to support HAWQ.

A DBMS can use Orca by implementing API to send catalog + stats + logical plans and then retrieve physical plans.

Supports multi-threaded search.
ORCA – ENGINEERING

Issue #1: Remote Debugging
→ Automatically dump the state of the optimizer (with inputs) whenever an error occurs.
→ The dump is enough to put the optimizer back in the exact same state later on for further debugging.

Issue #2: Optimizer Accuracy
→ Automatically check whether the ordering of the estimate cost of two plans matches their actual execution cost.
APACHE CALCITE

Standalone extensible query optimization framework for data processing systems.

→ Support for pluggable query languages, cost models, and rules.
→ Does not distinguish between logical and physical operators. Physical properties are provided as annotations.

Originally part of LucidDB.
MEMSQL OPTIMIZER

Rewriter
→ Logical-to-logical transformations with access to the cost-model.

Enumerator
→ Logical-to-physical transformations.
→ Mostly join ordering.

Planner
→ Convert physical plans back to SQL.
→ Contains MemSQL-specific commands for moving data.
MEMSQL OPTIMIZER OVERVIEW

SQL Query

Parser

Abstract Syntax Tree

Binder

Rewriter

Logical Plan

Physical Plan

Enumerator

Planner

Physical Plan

Cost Estimates

SQL
This is the part of a DBMS that I least understand. Let me know if you are interested in exploring this topic more.

All of this relies on a good cost model. A good cost model needs good statistics.
This is the part of a DBMS that I least understand. Let me know if you are interested in exploring this topic more.

All of this relies on a good **cost model**. A good **cost model** needs good statistics.
NEXT CLASS

Cost Models