Programmierpraktikum: Datensysteme
04 Background Query Optimization

Prof. Dr. Matthias Boehm
Technische Universität Berlin
Berlin Institute for the Foundations of Learning and Data
Big Data Engineering (DAMS Lab)

Last update: May 25, 2024

Announcements / Org

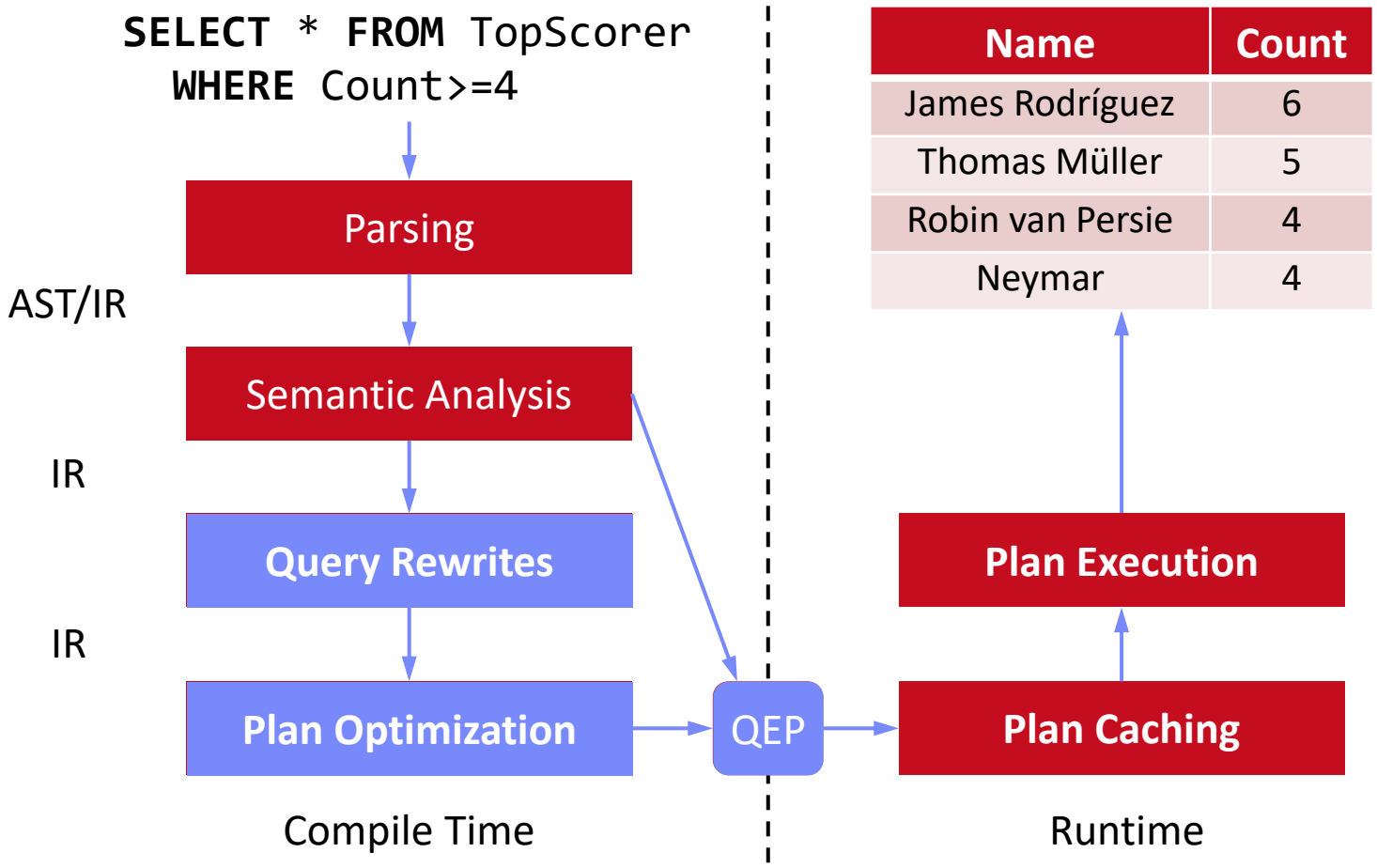
- **#1 Hybrid & Video Recording**
  - Hybrid lectures (in-person, zoom) with optional attendance
    https://tu-berlin.zoom.us/j/9529634787?pwd=R1ZsN1M3SC9BOU1OcFdmem9zT202UT09
  - Zoom **video recordings**, links from website
    https://mboehm7.github.io/teaching/ss24_ppds/index.htm

- **#2 Next Lecture**
  - **Jun 10 → 17 in MAR 0.003** (due to SIGMOD conference)
  - In-person instead of purely virtual

- **#3 Updated Reference Implementation**
  - Compilation issues w/ certain GCC versions
  - **Updated May 25**

[https://mboehm7.github.io/teaching/ss24_ppds/ppds_reference_cpp.zip]

Recap: Overview Query Processing



Name	Count
James Rodríguez	6
Thomas Müller	5
Robin van Persie	4
Neymar	4

Agenda

- **Query Rewriting and Unnesting**
- **Cardinality and Cost Estimation**
- **Join Enumeration / Ordering**

Query Rewriting and Unnesting

Query Rewrites

- **Query Rewriting**
  - Rewrite query into equivalent form that may be **processed more efficiently** or **give more freedom**
  - **#1 Same query can be expressed differently**, avoid hand-tuning
  - **#2 Complex queries may have redundancy**

- **A Simple Example**
  - Catalog meta data: custkey is unique

```
SELECT DISTINCT custkey, name
   FROM TPCH.Customer
```

⬇ rewrite

```
SELECT custkey, name
   FROM TPCH.Customer
```

- **20+ years of experience on query rewriting**

[Hamid Pirahesh, T. Y. Cliff Leung, Waqar Hasan: A Rule Engine for Query Transformation in Starburst and IBM DB2 C/S DBMS. **ICDE 1997**]

Standardization and Simplification

- **Normal Forms of Boolean Expressions**
  - **Conjunctive** normal form (P_{11} OR ... OR P_{1n}) **AND ... AND** (P_{m1} OR ... OR P_{mp})
  - **Disjunctive** normal form (P_{11} AND ... AND P_{1q}) **OR ... OR** (P_{r1} AND ... AND P_{rs})

- **Transformation Rules for Boolean Expressions**

Rule Name	Examples
Commutativity rules	A OR B ⇔ B OR A A AND B ⇔ B AND A
Associativity rules	(A OR B) OR C ⇔ A OR (B OR C) (A AND B) AND C ⇔ A AND (B AND C)
Distributivity rules	A OR (B AND C) ⇔ (A OR B) AND (A OR C) A AND (B OR C) ⇔ (A AND B) OR (A AND C)
De Morgan's rules	NOT (A AND B) ⇔ NOT (A) OR NOT (B) NOT (A OR B) ⇔ NOT (A) AND NOT (B)
Double-negation rules	NOT(NOT(A)) ⇔ A
Idempotence rules	A OR A ⇔ A A AND A ⇔ A A OR NOT(A) ⇔ TRUE A AND NOT (A) ⇔ FALSE A AND (A OR B) ⇔ A A OR (A AND B) ⇔ A A OR FALSE ⇔ A A OR TRUE ⇔ TRUE A AND FALSE ⇔ FALSE

Standardization and Simplification, cont.

- **Elimination of Common Subexpressions**
  - $(A_1=a_{11}$ **OR** $A_1=a_{12})$ **AND** $(A_1=a_{12}$ **OR** $A_1=a_{11}) \rightarrow A_1=a_{11}$ **OR** $A_1=a_{12}$
- **Propagation of Constants**
  - $A \geq B$ **AND** $B = 7 \rightarrow A \geq 7$ **AND** $B = 7$

$R \bowtie_{a=b} (\sigma_{b>0}(S)) \rightarrow (\sigma_{a>0}(R)) \bowtie_{a=b} (\sigma_{b>0}(S))$

- **Detection of Contradictions**
  - $A \geq B$ **AND** $B > C$ **AND** $C \geq A \rightarrow A > A \rightarrow$ **FALSE**
- **Use of Constraints**
  - A is primary key/unique: $\pi_A \rightarrow$ no duplicate elimination necessary
  - MAR_STATUS='married' → TAX_CLASS≥3: (MAR_STATUS='married' **AND** TAX_CLASS=1) → **FALSE**
- **Elimination of Redundancy** (set semantics)
  - $R \bowtie R \rightarrow R$, $R \cup R \rightarrow R$, $R - R \rightarrow \emptyset$
  - $R \bowtie (\sigma_p R) \rightarrow \sigma_p R$, $R \cup (\sigma_p R) \rightarrow R$, $R - (\sigma_p R) \rightarrow \sigma_{\neg p} R$
  - $(\sigma_{p1} R) \bowtie (\sigma_{p2} R) \rightarrow \sigma_{p1 \wedge p2} R$, $(\sigma_{p1} R) \cup (\sigma_{p2} R) \rightarrow \sigma_{p1 \vee p2} R$

Query Unnesting

[Won Kim: On Optimizing an SQL-like Nested Query. **ACM Trans. Database Syst. 1982**]

- **Case 1: Type-A Nesting**
  - Inner block is not correlated and computes an aggregate
  - **Solution:** Compute the aggregate once and insert into outer query

```
SELECT OrderNo FROM Order
  WHERE ProdNo =
   (SELECT MAX(ProdNo)
      FROM Product WHERE Price<100)
```

```
$X = SELECT MAX(ProdNo)
   FROM Product WHERE Price<100

SELECT OrderNo FROM Order
  WHERE ProdNo = $X
```

- **Case 2: Type-N Nesting**
  - Inner block is not correlated and returns a set of tuples
  - **Solution:** Transform into a symmetric form (via join)

```
SELECT OrderNo FROM Order
  WHERE ProdNo IN
   (SELECT ProdNo
      FROM Product WHERE Price<100)
```

```
SELECT OrderNo
 FROM Order O, Product P
 WHERE O.ProdNo = P.ProdNo
   AND P.Price < 100
```

Query Unnesting, cont.

[Won Kim: On Optimizing an SQL-like Nested Query. **ACM Trans. Database Syst. 1982**]

- **Case 3: Type-J Nesting**
  - Un-nesting of correlated sub-queries w/o aggregation

```
SELECT OrderNo FROM Order O
  WHERE ProdNo IN
   (SELECT ProdNo FROM Project P
    WHERE P.ProjNo = O.OrderNo
      AND P.Budget > 100,000)
```

```
SELECT OrderNo
  FROM Order O, Project P
  WHERE O.ProdNo = P.ProdNo
    AND P.ProjNo = O.OrderNo
    AND P.Budget > 100,000
```

- **Case 4: Type-JA Nesting**
  - Un-nesting of correlated sub-queries w/ aggregation
  - Further un-nesting via case 3 and 2

```
SELECT OrderNo FROM Order O
  WHERE ProdNo IN
   (SELECT MAX(ProdNo)
     FROM Project P
     WHERE P.ProjNo = O.OrderNo
       AND P.Budget > 100,000)
```

```
SELECT OrderNo FROM Order O
  WHERE ProdNo IN
   (SELECT ProdNo FROM
     (SELECT ProjNo, MAX(ProdNo)
       FROM Project
       WHERE Budget > 100.000
       GROUP BY ProjNo) P
    WHERE P.ProjNo = O.OrderNo)
```

Unnesting Arbitrary Queries

- **Overview**
  - General transformation for elimination of **dependent joins**
  - Guaranteed lower or equal cost / reuse of subsequent rewrites

[Thomas Neumann, Alfons Kemper: Unnesting Arbitrary Queries. **BTW 2015**]

- **#1 Simple Unnesting**
  - Move dependent predicates up as far as possible
  - Transforms dependent into regular join if adjacent

- **#2 General Unnesting**
  - Translate dependent join into regular and **deduplicated dependent join**

$$T_1 \bowtie_p T_2 \quad \equiv \quad T_1 \bowtie_{p \wedge T_1 =_{\mathcal{A}(D)} D} (D \bowtie T_2)$$

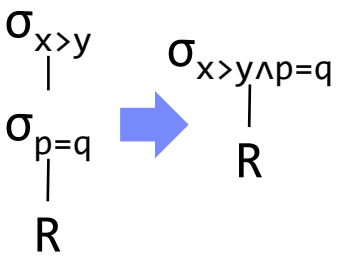
$$D := \Pi_{\mathcal{F}(T_2) \cap \mathcal{A}(T_1)}(T_1)$$

  - Push down dependent join, turn dependent join over base relation into **regular join**
  - Specific optimizations (e.g., **sideways information passing**), other rewrites

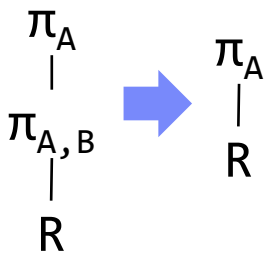
Selections and Projections

- **Example Transformation Rules**

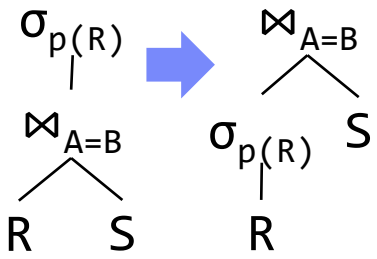
1) Grouping of Selections



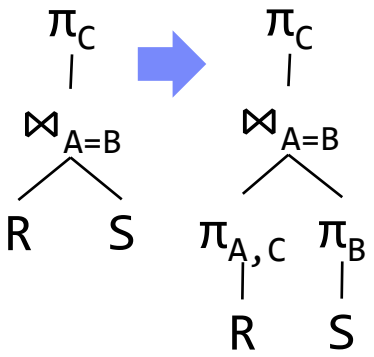
2) Grouping of Projections



3) Pushdown of Selections



4) Pushdown of Projections



- **Restructuring Algorithm**
  - **#1** Split n-ary joins into binary joins
  - **#2** Split multi-term selections
  - **#3** Push-down selections as far as possible
  - **#4** Group adjacent selections again
  - **#5** Push-down projections as far as possible

Input: Standardized, simplified, and un-nested query graph

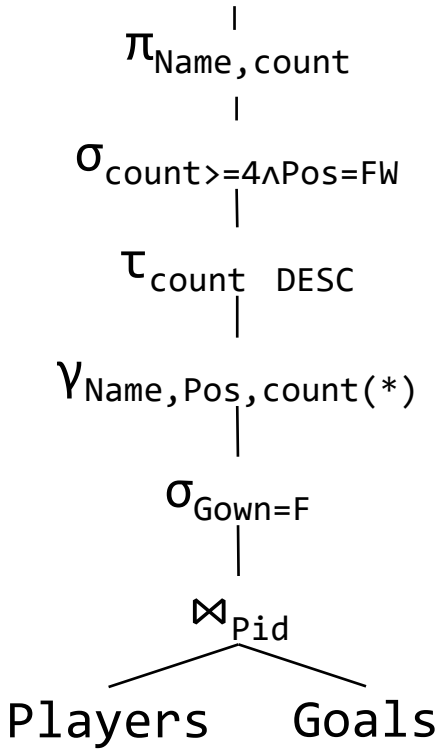
Output: Restructured query graph

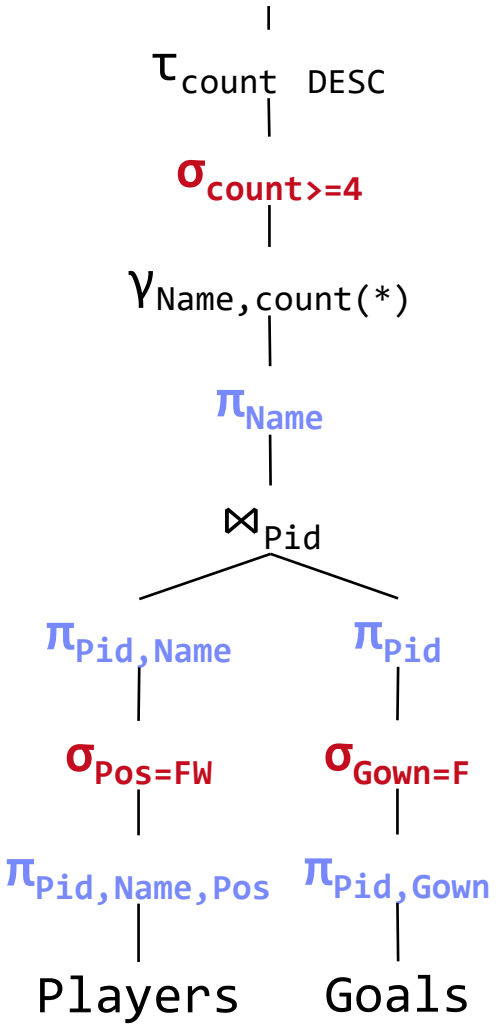
Example Query Restructuring

```
SELECT Name, count
  FROM TopScorer
  WHERE count>=4
  AND Pos='FW'
```

```
CREATE VIEW TopScorer AS
SELECT P.Name, P.Pos, count(*)
  FROM Players P, Goals G
  WHERE P.Pid=G.Pid
    AND G.GOwn=FALSE
  GROUP BY P.Name, P.Pos
  ORDER BY count(*) DESC
```

Additional metadata:
- P.Name is unique





Cardinality and Cost Estimation

Overview Cost Models

[Guido Moerkotte, Building Query Compilers (Under Construction), **2020**, http://pi3.informatik.uni-mannheim.de/~moer/querycompiler.pdf]

- **Overall Cost Models**
  - **I/O costs** (number of read pages, tuples)
  - **Computation costs** (CPU costs, tuples)
  - Others: Memory, Energy
  - Aggregate operator costs (specific vs general) w/ awareness of parallelism

$$C = C_{I/O} + C_{CPU}$$

$$C = \max(C_{I/O}, C_{CPU})$$

- **Cost Model Inputs**
  - Base relations: number of pages, number of tuples, avg tuple length
  - Intermediates: number of tuples → **Cardinality estimation**

- **Common Assumptions**
  - **No Skew:** uniform value distributions of attributes
  - **Independence:** no correlation among attributes
    → underestimation → poor plans

	(estimated)	**(real)**
$\sigma_{\text{Model= 'Golf'}}$	10	**590**
$\sigma_{\text{Make='VW'}}$	1,000	**5,000**
Cars	10,000	10,000

Cardinality and Selectivity

[Guido Moerkotte, Building Query Compilers, **2020**]

- **Cardinality** $|R|$
  - Size of intermediates in number of tuples (sometimes distinct items)
  - **Examples:** $|\sigma_p R|$, $|R \bowtie S|$

- **Selectivity** $s(p)$
  - Fraction of tuples that pass operator, bounded by [0,1]
  - **"Highly-selective"** operator → low selectivity $s(p)$

  - **Example Selection**

$$s(p) = \frac{|\sigma_p R|}{|R|} \quad \Rightarrow \quad |\sigma_p R| = s(p) \cdot |R|$$

  - **Example Join**

$$s(p) = \frac{|R \bowtie_p S|}{|R \times S|} = \frac{|R \bowtie_p S|}{|R| \cdot |S|} \quad \Rightarrow \quad |R \bowtie_p S| = s(p) \cdot |R| \cdot |S|$$

Cardinality Propagation

[Guido Moerkotte, Building Query Compilers, **2020**]

- **Operator-level Propagation**
  - Selection: $|\sigma_p R| = s(p) \cdot |R|$
  - Join: $|R \bowtie_p S| = s(p) \cdot |R| \cdot |S|$
  - Sorting: $|\tau_A(R)| = |R|$
  - Group-by: $|\gamma_{G;f}(R)| = \prod_{g \in G} d_g(R)$
  - Cross product: $|R \times S| = |R| \cdot |S|$
  - Projection: $|\pi(R)| = |R|$
  - Union All: $|R \cup S| = |R| + |S|$

→ **Recursive propagation over query tree**

- **Error Propagation**
  - Cardinality estimation errors propagate **exponentially through joins** (max error)

[Yannis E. Ioannidis, Stavros Christodoulakis: On the Propagation of Errors in the Size of Join Results. **SIGMOD 1991**]

- **Q-Error**
  - **Multiplicative error**, produced plans at most q^4 worse than optimum

[Guido Moerkotte, Thomas Neumann, Gabriele Steidl: Preventing Bad Plans by Bounding the Impact of Cardinality Estimation Errors. **PVLDB 2(1) 2009**]

Cardinality Propagation, cont.

[Patricia G. Selinger et al.: Access Path Selection in a Relational Database Management System. **SIGMOD 1979**]

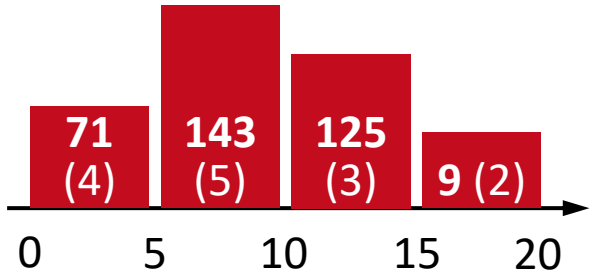
- **Equality Predicates**
  - Based on histograms and #distinct item estimators, otherwise default 1/10
  - Constant predicate: $s(A = c) = \frac{1}{d_A}$ //assumes uniformity
  - Binary predicate: $s(A = B) = \frac{1}{\max(d_A, d_B)}$ //assumes matching domains
- **Range Predicates**
  - One-sided: $s(A > c) = \frac{\max_A - c}{\max_A - \min_A}$
  - Two-sided: $s(c_1 \leq A \leq c\_2) = \frac{c_2 - c_1}{\max_A - \min_A}$
- **Composite Predicates** (→ sparsity in ML systems)
  - Negation (NOT): $s(\neg p) = 1 - s(p)$
  - Conjunction (AND): $s(p_1 \wedge p_2) = s(p_1) \cdot s(p_2)$ //assumes independence
  - Disjunction (OR): $s(p_1 \vee p_2) = s(p_1) + s(p_2) - s(p_1) \cdot s(p_2)$

Cardinality Estimation

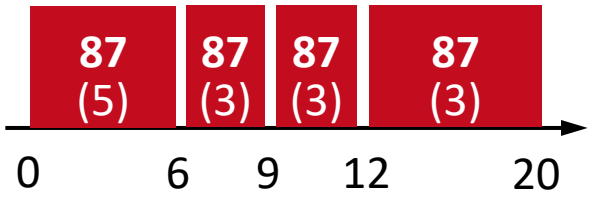
[Guido Moerkotte, Building Query Compilers, **2020**]

- **Overview**
  - Min, Max, #distinct items d crucial for cardinality estimation
  - Exact frequency distribution $(v_1, f_1), (v_2, f_2), \ldots, (v_d, f_d)$ too detailed

- **Equi-width Histogram**
  - Divide min-max range into B buckets
  - Store sum frequency, #distinct



- **Equi-height Histogram**
  - Divide range into variable buckets with constant frequency
  - E.g., via quantiles + duplicate handling



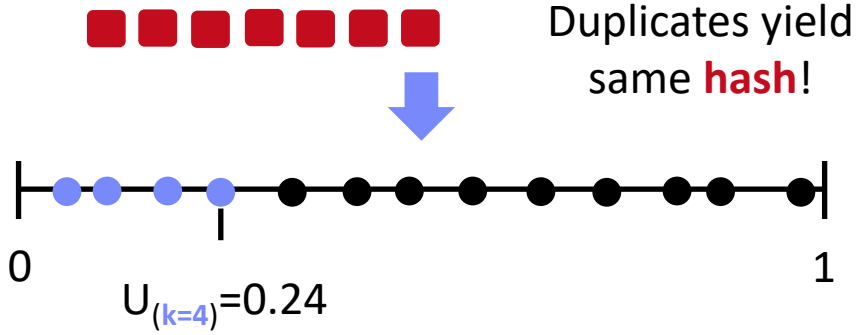
[Carl-Christian Kanne, Guido Moer-kotte: Histograms reloaded: the merits of bucket diversity. **SIGMOD 2010**]

- **Other Histograms**
  - Homogeneous/heterogeneous histograms w/ bounded error

Number of Distinct Items

- **Problem**
  - **Estimate # distinct items** in a dataset / data stream w/ limited memory
  - Support for set operations (union, intersect, difference)

- **K-Minimum Values (KMV)**
  - Hash values d_i to $h_i \in [0, M]$ with domain $M = O(D^2)$ to avoid collisions → **$O(k \log D)$ space**
  - **Store k minimum hash values** (e.g., via priority queue) in normalized form $h_i \in [0,1]$



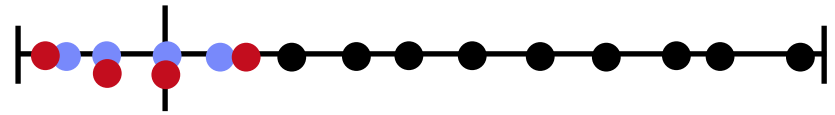
  - Basic estimator: $\hat{D}_k^{BE} = k/U_{(k)}$
  - **Unbiased estimator:** $\hat{D}_k^{UB} = (k-1)/U_{(k)}$

Example:
16.67 vs 12.5

[Kevin S. Beyer, Peter J. Haas, Berthold Reinwald, Yannis Sismanis, Rainer Gemulla: On synopses for distinct-value estimation under multiset operations. **SIGMOD 2007**]

Number of Distinct Items, cont.

- **KMV Set Operations**
  - Union/intersection directly on partition synopses
  - Difference via **Augmented KMV** (AKMV) that include counters of multiplicities of k-minimum values



$$D = A \cup B$$

$$KMV(D_{\cup}) \equiv KMV(A) \oplus KMV(B)$$

- **HyperLogLog**
  - Hash values and maintain maximum **# of leading zeros** p → $\hat{D} = 2^p$
  - Stochastic averaging over M streams (p maintained in M registers)
  - **HyperLogLog++**
  - Updatable HyperLogLog, with sampling for multi-column estimates

[P. Flajolet, Éric Fusy, O. Gandouet, and F. Meunier: Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. **AOFA 2007**]

[Stefan Heule, Marc Nunkesser, Alexander Hall: HyperLogLog in practice: algorithmic engineering of a state of the art cardinality estimation algorithm. **EDBT 2013**]

[Michael J. Freitag, Thomas Neumann: Every Row Counts: Combining Sketches and Sampling for Accurate Group-By Result Estimates. **CIDR 2019**]

Sample-based Cardinality Estimation

- **Overview and Problems**
  - Sample subset S with $|S| \ll N$ of tuples and estimate #distinct items d
  - Naïve estimators: d_S → **underestimate**, or $d_S \cdot N/|S|$ → **overestimate**

- **#1 Sample-based Estimators**
  - "Generalized jackknife" estimator

[P. J. Haas and L. Stokes: Estimating the Number of Classes in a Finite Population, **J. Amer. Statist. Assoc., 93(444), 1998**]

squared coefficient of variation

simple estimator $\hat{d}_{\mathrm{uj1}} = \left(1 - (1-q)(h_1/|\mathcal{S}|)\right)^{-1} d_{\mathcal{S}}$

$$\hat{d}_{hybrid} = \begin{cases} \hat{d}_{uj2}, & 0 < \hat{\gamma}^2(\hat{d}_{uj1}) < \alpha_1 \\ \hat{d}_{uj2a}, & \alpha_1 \leq \hat{\gamma}^2(\hat{d}_{uj1}) < \alpha_2 \\ \hat{d}_{Sh3}, & otherwise \end{cases}$$

$$\hat{d} = d_S + K \cdot f_1/N$$

  - Guaranteed error estimator (GEE)
    - Basic and adaptive estimators

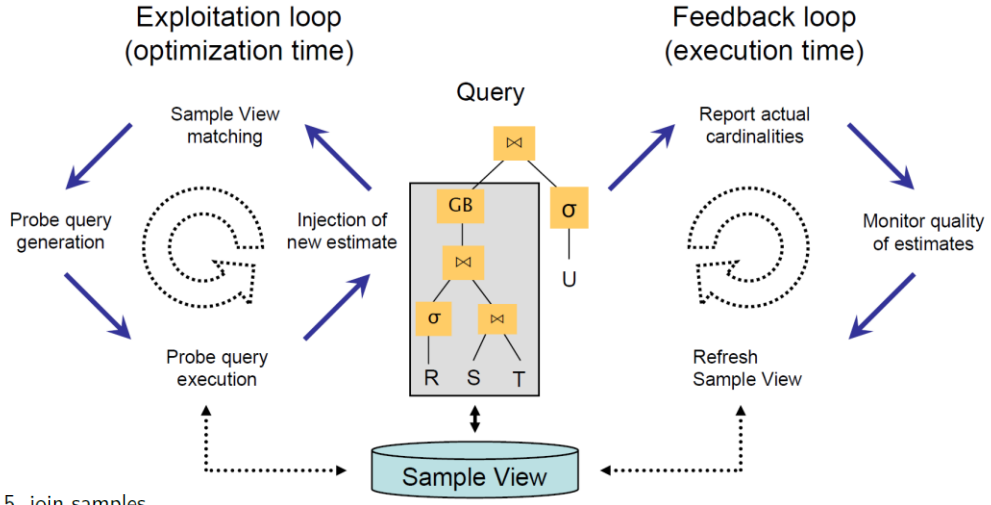
[Moses Charikar, Surajit Chaudhuri, Rajeev Motwani, Vivek R. Narasayya: Towards Estimation Error Guarantees for Distinct Values. **PODS 2000**]

$$\hat{d} = \sqrt{\frac{N}{|S|}} f_1 + \sum_{i=2}^{|S|} f_i$$

Sample-based Cardinality Estimation, cont.

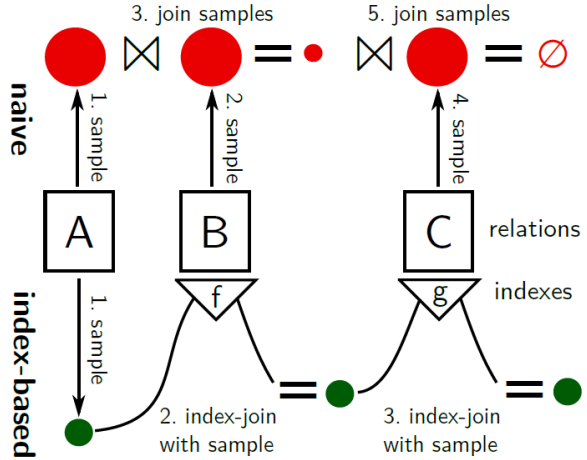
- **#2 Sample Views**
  - Random sampling + materialized views w/ statistical guarantees
  - Query feedback (actual card)

[Per-Åke Larson, Wolfgang Lehner, Jingren Zhou, Peter Zabback: Cardinality estimation using sample views with quality assurance. **SIGMOD 2007**]

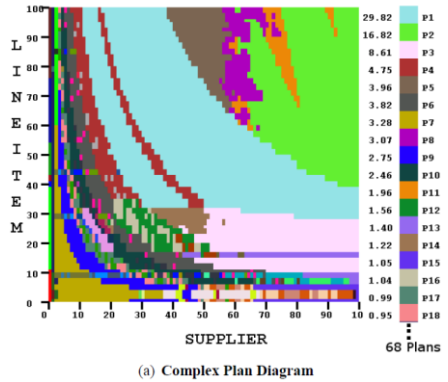


- **#3 Index-based Join Sampling**
  - Joins on samples might result in ∅
  - Use existing indexes to explore intermediate results bottom-up

[Viktor Leis, Bernhard Radke, Andrey Gubichev, Alfons Kemper, Thomas Neumann: Cardinality Estimation Done Right: Index-Based Join Sampling. **CIDR 2017**]



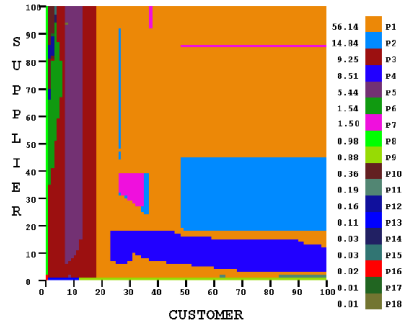
Excursus: Robust Query Optimization



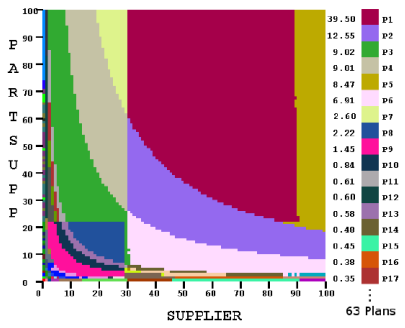
(a) Complex Plan Diagram

- **Overview Picasso Project**
  - **Plan diagram:** plan choice over selectivity ranges
  - **Cost diagram:** estimated plan execution costs over ranges

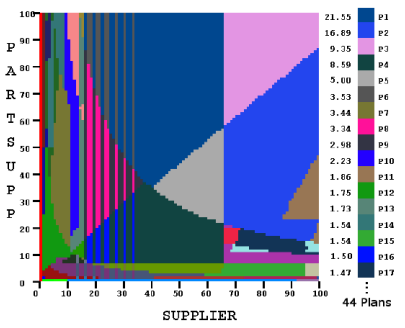
Duplicate Islands



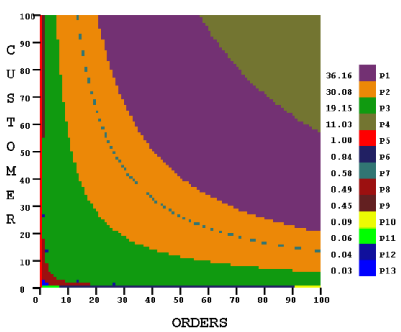
Plan Switch Points



Venetian Blinds

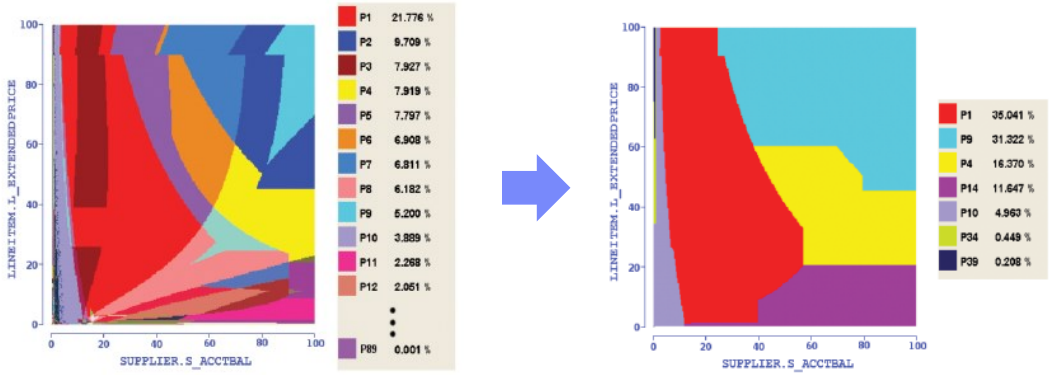


Footprint Pattern



- **Towards Robust Optimization**

[Naveen Reddy, Jayant R. Haritsa: Analyzing Plan Diagrams of Database Query Optimizers. **VLDB 2005**]



Excursus: Robust Query Optimization, cont.

[Harish Doraiswamy, Pooja N. Darera, Jayant R. Haritsa:
On the Production of Anorexic Plan Diagrams. **VLDB 2007**]

[Harish Doraiswamy, Pooja N. Darera, Jayant R. Haritsa:
Identifying robust plans through plan diagram reduction. **PVLDB 1(1) 2008**]

[M. Abhirama, Sourjya Bhaumik, Atreyee Dey, Harsh Shrimal, Jayant R. Haritsa:
On the Stability of Plan Costs and the Costs of Plan Stability. **PVLDB 3(1) 2010**]

[Goetz Graefe, Wey Guy, Harumi A. Kuno, Glenn N. Paulley:
Robust Query Processing (Dagstuhl Seminar 12321). **Dagstuhl Reports 2(8) 2012**]

[Anshuman Dutt, Jayant R. Haritsa:
Plan bouquets: query processing without selectivity estimation. **SIGMOD 2014**]

[Jayant R. Haritsa: Robust Query Processing:
Mission Possible. **PVLDB 13(12) 2020**]

Adaptive Query Processing
(learned cardinalities, re-optimization)

Join Enumeration / Ordering

Plan Optimization Overview

[Guido Moerkotte, Building Query Compilers, **2020**]

- **Plan Generation Overview**
  - Selection of **physical access path and plan operators**
  - Selection of **execution order** of plan operators (**joins**, group-by)
  - **Input:** logical query plan → **Output:** optimal physical query plan
  - Costs of query optimization should not exceed yielded improvements

- **Interesting Properties**
  - Interesting orders (sorted vs unsorted), partitioning (e.g., join column), pipelining
  - Avoid unnecessary sorting operations

[Ihab F. Ilyas, Jun Rao, Guy M. Lohman, Dengfeng Gao, Eileen Tien Lin: Estimating Compilation Time of a Query Optimizer. **SIGMOD 2003**]

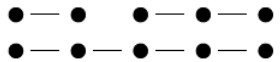
- **Simple Cost Functions**
  - Join-specific cost functions (Cnlj, Chj, Csmj)
  - Cardinalities Cout

$$C_{\text{out}}(T) = \begin{cases} 0 & \text{if } T \text{ is a single relation} \\ |T| + C_{\text{out}}(T_1) + C_{\text{out}}(T_2) & \text{if } T = T_1 \bowtie T_2 \end{cases}$$

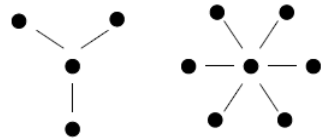
Query and Plan Types

[Guido Moerkotte, Building Query Compilers, **2020**]

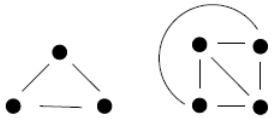
- **Query Types**
  - **Nodes:** Tables
  - **Edges:** Join conditions
  - Determine **hardness of query optimization** (w/o cross products)



Chains



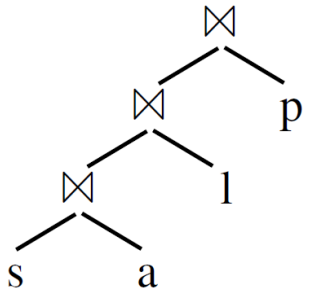
Stars



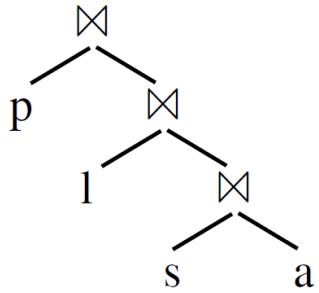
Cliques

- **Join Tree Types / Plan Types**
  - Data flow graph of tables and joins (logical/physical query trees)
  - **Edges:** data dependencies (execution order: bottom-up)

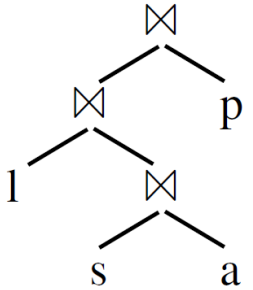
Left-Deep Tree



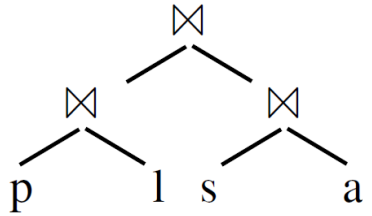
Right-Deep Tree



Zig-Zag Tree



Bushy Tree



Join Ordering Problem

[Guido Moerkotte, Building Query Compilers, **2020**]

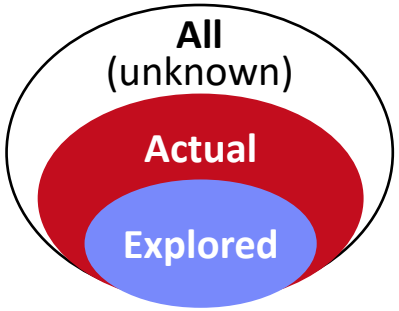
- **Join Ordering**
  - Given a join query graph, find the optimal join ordering
  - In general, **NP-hard**; but polynomial algorithms exist for special cases

- **Search Space**
  - Dependent on query & plan types
  - **Note:** if we allow cross products similar to cliques (fully connected)

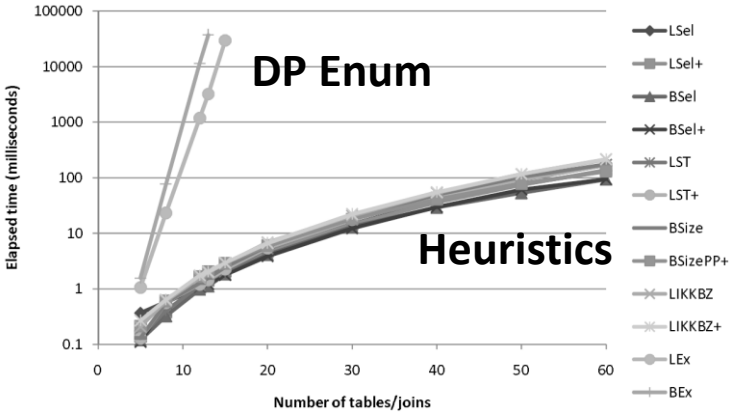
	Chain (no CP)			Star (no CP)		Clique / CP (cross product)		
	left-deep	**zig-zag**	**bushy**	**left-deep**	**zig-zag/ bushy**	**left-deep**	**zig-zag**	**bushy**
n	2^{n-1}	2^{2n-3}	$2^{n-1}C(n-1)$	$2(n-1)!$	$2^{n-1}(n-1)!$	$n!$	$2^{n-2}n!$	$n!\ C(n-1)$
5	16	128	224	48	384	120	960	1,680
10	512	~131K	~2.4M	~726K	~186M	~3.6M	~929M	~17.6G

C(n) … Catalan Numbers

Join Order Search Strategies



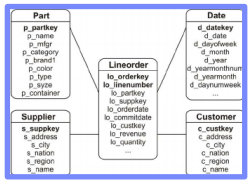
- **Tradeoff: Optimal (or good)** plan vs **compilation time**
- **#1 Naïve Full Enumeration**
  - Infeasible for reasonably large queries (long tail up to 1000s of joins)
- **#2 Exact Dynamic Programming / Memoization**
  - Guarantees optimal plan, often too expensive (beyond 20 relations)
  - Bottom-up vs top-down approaches
- **#3 Greedy / Heuristic Algorithms**
- **#4 Approximate Algorithms**
  - E.g., Genetic algorithms, simulated annealing, MIL programming
- **Example PostgreSQL**
  - Exact optimization (DPSize) if **< 12 relations** (geqo_threshold)
  - Genetic algorithm for larger queries
  - Join methods: NLJ, SMJ, HJ



[Nicolas Bruno, César A. Galindo-Legaria, Milind Joshi: Polynomial heuristics for query optimization. **ICDE 2010**]

Greedy Join Ordering

Star Schema Benchmark



- **Example**
  - Part ⋈ Lineorder ⋈ Supplier ⋈ σ(Customer) ⋈ σ(Date), **left-deep plans**

#	Plan	Costs
1	Lineorder ⋈ Part	30M
	Lineorder ⋈ Supplier	20M
	Lineorder ⋈ σ(Customer)	90K
	Lineorder ⋈ σ(Date)	**40K**
	~~Part ⋈ Customer~~	N/A
	…	…
2	(Lineorder ⋈ σ(Date)) ⋈ Part	150K
	(Lineorder ⋈ σ(Date)) ⋈ Supplier	100K
	(Lineorder ⋈ σ(Date)) ⋈ σ(Customer)	**75K**
3	((Lineorder ⋈ σ(Date)) ⋈ σ(Customer)) ⋈ Part	120M
	((Lineorder ⋈ σ(Date)) ⋈ σ(Customer)) ⋈ Supplier	**105M**
4	**(((Lineorder ⋈ σ(Date)) ⋈ σ(Customer)) ⋈ Supplier) ⋈ Part**	**135M**

Note: Simple $O(n^2)$ algorithm for left-deep trees; $O(n^3)$ algorithms for bushy trees existing (e.g., GOO .. Greedy operator ordering)

Greedy Join Ordering, cont.

[Guido Moerkotte, Building Query Compilers, **2020**]

- **Basic Algorithms**
  - GreedyJO-1: sort by relation weights (e.g., card)
  - GreedyJO-2: greedy selection of next best relation
  - GreedyJO-3: Greedy-JO-2 w/ start from each relation

  } Previous example as a hybrid w/ $O(n^2)$

- **GOO Algorithm**

```
GOO({R_1, ..., R_n})   // Greedy Operator Ordering
Input:  a set of relations to be joined
Output:  join tree
Trees := {R_1, ..., R_n}
while (|Trees| != 1) {
  find T_i, T_j ∈ Trees such that i ≠ j, |T_i ⋈ T_j| is minimal
       among all pairs of trees in Trees
  Trees -= T_i;
  Trees -= T_j;
  Trees += T_i ⋈ T_j;
}
return the tree contained in Trees;
```

[Leonidas Fegaras: A New Heuristic for Optimizing Large Queries. **DEXA 1998**]

Dynamic Programming Join Ordering

- **Exact Enumeration via Dynamic Programming**
  - **#1:** Optimal substructure (Bellman's Principle of Optimality)
  - **#2:** Overlapping subproblems allow for memorization

- **Bottom-Up** (Dynamic Programming)
  - Split in independent sub-problems (optimal plan per set of quantifiers and interesting properties), solve sub-problems, combine solutions
  - **Algorithms:** DPsize, DPsub, DPcpp

- **Top-Down** (Memoization)
  - Recursive generation of join trees w/ memorization and pruning
  - **Algorithms:** Cascades, MinCutLazy, MinCutAGat, MinCutBranch

[Guido Moerkotte, Thomas Neumann: Analysis of Two Existing and One New Dynamic Programming Algorithm for the Generation of Optimal Bushy Join Trees without Cross Products. **VLDB 2006**]

[Goetz Graefe: The Cascades Framework for Query Optimization. **IEEE Data Eng. Bull. 18(3) 1995**]

[Pit Fender: Algorithms for Efficient Top-Down Join Enumeration. **PhD Thesis, University of Mannheim 2014**]

Dynamic Programming Join Ordering, cont.

[Patricia G. Selinger et al.: Access Path Selection in a Relational Database Management System. **SIGMOD 1979**]

- **DPSize Algorithm**
  - Pioneered by Pat Selinger et al.
  - Implemented in IBM DB2, Postgres, etc

Algorithm 1 SerialDPEnum

Input: a connected query graph with quantifiers $q_1, \cdots, q_N$
Output: an optimal bushy join tree

```
1:  for i ← 1 to N
2:    Memo[{q_i}] ← CreateTableAccessPlans(q_i);
3:    PrunePlans(Memo[{q_i}]);
4:  for S ← 2 to N
5:    for smallSZ ← 1 to ⌊S/2⌋
6:      largeSZ ← S − smallSZ;
7:      for each smallQS of size smallSZ
8:        for each largeQS of size largeSZ
9:          if smallQS ∩ largeQS ≠ ∅ then
10:           continue; /*discarded by the disjoint filter*/
11:         if not(smallQS connected to largeQS) then
12:           continue; /*discarded by the connectivity filter*/
13:         ResultingPlans ← CreateJoinPlans(
                Memo[smallQS], Memo[largeQS]);
14:       PrunePlans(Memo[smallQS ∪ largeQS], ResultingPlans);
15: return Memo[{q_1, ··· , q_N}];
```

disjoint

connected

[Wook-Shin Han, Wooseong Kwak, Jinsoo Lee, Guy M. Lohman, Volker Markl: Parallelizing query optimization. **PVLDB 1(1) 2008**]

Dynamic Programming Join Ordering, cont.

- **DPSize Example**
  - Simplified: no interesting properties

Q1	Plan
{C}	Tbl, ~~IX~~
{D}	~~Tbl~~, IX
{L}	…
{P}	…
{S}	…

Q1+Q1

Q2	Plan
{C,L}	L⋈C, ~~C⋈L~~
{D,L}	L⋈D, ~~D⋈L~~
{L,P}	~~L⋈P~~, P⋈L
{L,S}	~~L⋈S~~, S⋈L
~~{C,D}~~	~~N/A~~
…	…

Q1+Q2, Q2+Q1

Q3	Plan
{C,D,L}	(L⋈C)⋈D, ~~D⋈(L⋈C)~~, ~~(L⋈D)⋈C~~, ~~C⋈(L⋈D)~~
{C,L,P}	~~(L⋈C)⋈P~~, P⋈(L⋈C), ~~(P⋈L)⋈C~~, ~~C⋈(P⋈L)~~
{C,L,S}	…
{D,L,P}	…
{D,L,S}	…
{L,P,S}	…

Q1+Q3, Q2+Q2, Q3+Q1

Q4	Plan
{C,D,L,P}	~~((L⋈C)⋈D)⋈P~~, P⋈((L⋈C)⋈D)
{C,D,L,S}	…
{C,L,P,S}	…
{D,L,P,S}	…

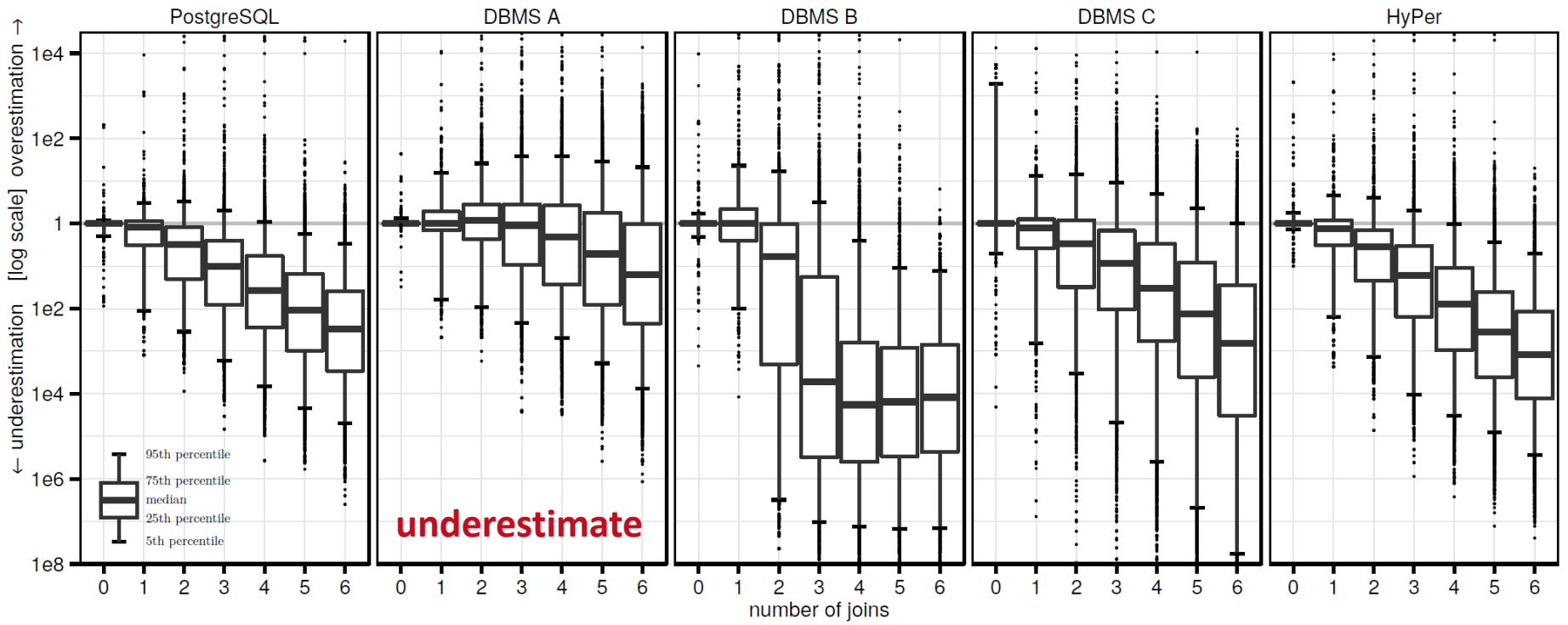
Q1+Q4, Q2+Q3, Q3+Q2, Q4+Q1

Q5	Plan
{C,D,L,P,S}	…

Join Order Benchmark (JOB)

[Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter A. Boncz, Alfons Kemper, Thomas Neumann: How Good Are Query Optimizers, Really? **PVLDB 9(3) 2015**]

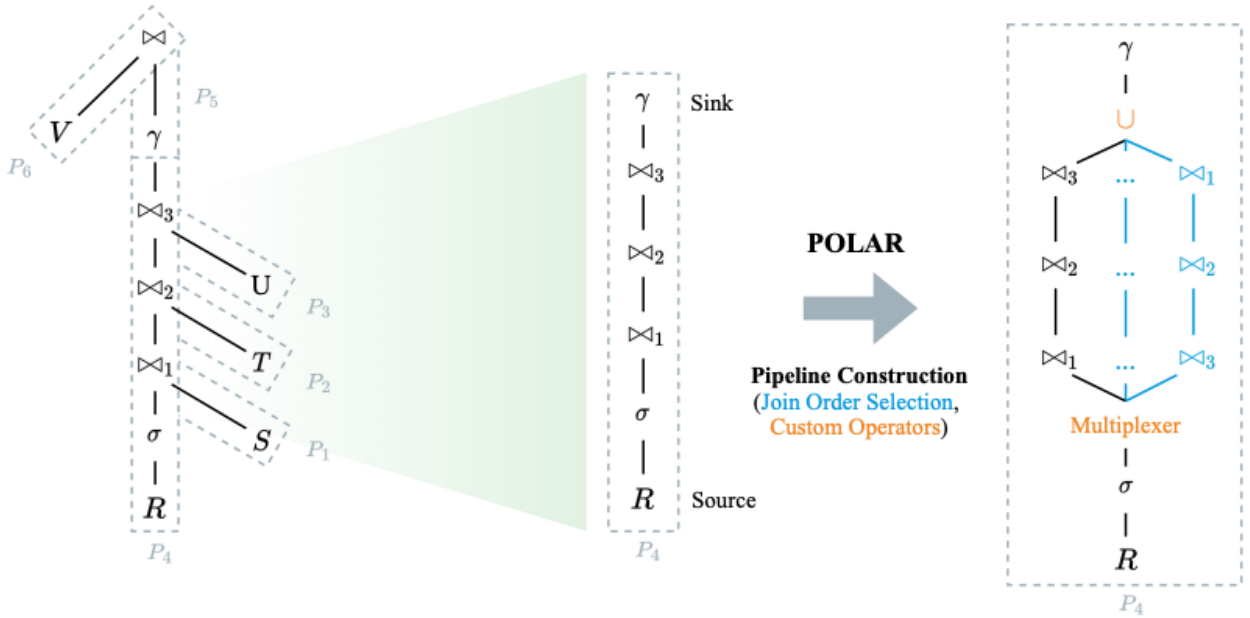
- **Data:** Internet Movie Data Bases (IMDB)
- **Workload:** 33 query templates, 2-6 variants / 3-16 joins per query

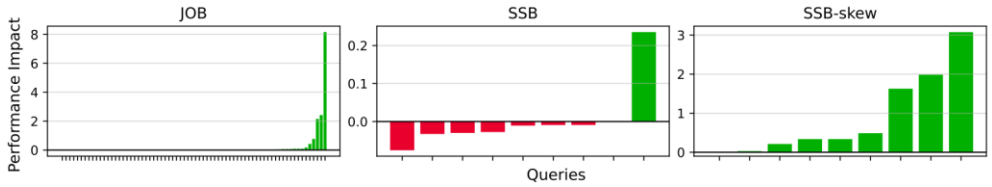


Excursus: POLAR: Non-invasive, Low Overhead Adaptive Query Processing [DIMA/DAMS @ VLDB'24]

→ BIFOLD GS (DAMS/DIMA)

- **Simple Integration and Low Overhead**
  - Separate compilation and execution
  - Reuse optimizers and operators
- **Join Order Selection**
  - Discretize selectivities (predicates and joins)
  - Sample the selectivity space
  - Optimize plan for sample, include original plan
- **Determine Plans of Least Resistance**
  - Measure # intermediates during runtime
  - Explore join plans via tuple routing w/ bounded regret
- **Next Steps**
  - DuckDB extension in DuckDB OSS repo (BS Paul Pohlitz)
  - Skewed data generation (BS Marcel Scholand)





Summary & QA

- **Query Rewriting and Unnesting**
- **Cardinality and Cost Estimation**
- **Join Enumeration / Ordering**

Thanks

- **Next Lectures**
  - **05 Experiments and Reproducibility** [Jun 10 → **Jun 17 in MAR 0.003**]
  - **Project Submissions** [Jul 01 EOD]
  - **Project Presentations all Teams** [Jul 08]