On the Performance of Query Optimization Without Cost Functions and Very Simple Cardinality Estimation

DBKDA @ InfoSys 2025, Lisbon, Portugal

Daniel Flachs and Guido Moerkotte

Database Research Group, University of Mannheim, Germany

{flachs,moerkotte}@uni-mannheim.de

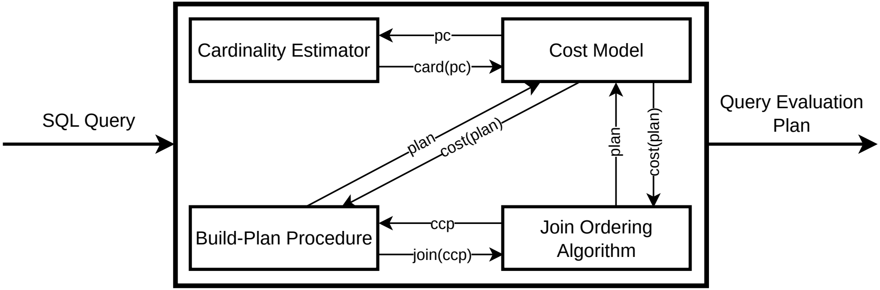
March 11, 2025

Daniel Flachs is a PhD student at the Database Research Group at the University of Mannheim, Germany. His research interest involve traditional query optimization and query processing in relational database management systems, focusing on the interaction of join implementations, cost models, and cardinality estimation.

Guido Moerkotte is a professor for computer science at the University of Mannheim, Germany, and head of the Database Research Group. His research is focused on all aspects of query optimization for relational database management systems.

Introduction and Motivation

Modules of a **Database Management System (DBMS)** w.r.t. **Query Optimization (QO)**:



When developing a new DBMS, implementing these requires **compromise**.

Introduction and Motivation

Recent attempts to **query optimization simplification**:

- Hertzschuch et al. (2021)
- Datta et al. (2024)

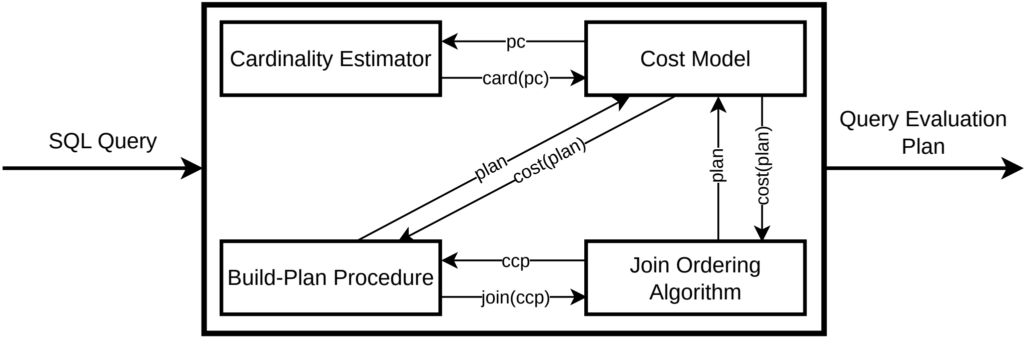
Research Questions

1. How simple can cardinality estimation (CE), cost function (CF), and build-plan procedure (BP) become while still yielding QEPs with an acceptable quality?
2. Can we get acceptable plans even if BP does not use any cost function at all?

Contributions

- Two new simple cardinality estimators: $\mathrm{CE}_{\mathrm{base}}$ + $\mathrm{CE}_{\mathrm{sel}}$
- New build-plan procedure without cost function: $\mathrm{BP}_{\mathrm{smart}}$
- Two cost functions of different precision for two hash join implementations: $\mathrm{CF}_{\mathrm{tru}}$ + $\mathrm{CF}_{\mathrm{est}}$

Preliminaries: Overview



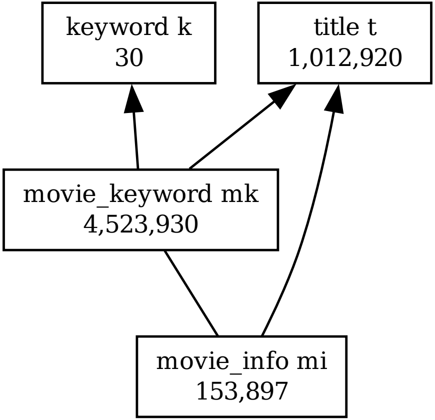
Query Optimizer Architecture

Queries and Query Graphs

SQL Query (JOB Query 3a)

```
SELECT MIN(t.title) AS movie_title
FROM keyword AS k,
     movie_info AS mi,
     movie_keyword AS mk,
     title AS t
WHERE /* some selection predicates */
      AND t.id = mi.movie_id
      AND t.id = mk.movie_id
      AND mk.movie_id = mi.movie_id
      AND k.id = mk.keyword_id;
```

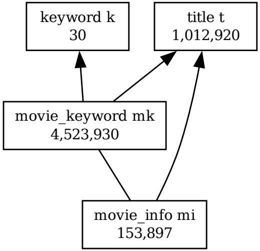
Query Graph



Plan Classes and Cardinality Estimation

Plan class (PC): subset of relations that induces a connected subgraph of the query graph.

- PCs of size 1: $\{\mathtt{k}\}, \{\mathtt{t}\}, \{\mathtt{mk}\}, \{\mathtt{mi}\}$
- PCs of size 2: $\{\mathtt{k}, \mathtt{mk}\}, \{\mathtt{t}, \mathtt{mk}\}, \{\mathtt{t}, \mathtt{mi}\}, \{\mathtt{mk}, \mathtt{mi}\}$
- PCs of size 3: $\{\mathtt{k}, \mathtt{t}, \mathtt{mk}\}, \{\mathtt{k}, \mathtt{mk}, \mathtt{mi}\}, \{\mathtt{t}, \mathtt{mk}, \mathtt{mi}\}$
- PCs of size 4: $\{\mathtt{k}, \mathtt{t}, \mathtt{mk}, \mathtt{mi}\}$

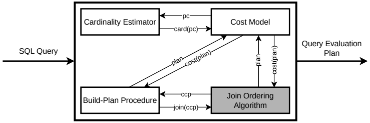


There are many different logical/physical QEPs for the relations of the same plan class.

Observe that all plans of the same plan class have the same result cardinality: $|(\mathtt{k} \bowtie \mathtt{mk}) \bowtie \mathtt{t}| = |(\mathtt{mk} \bowtie \mathtt{t}) \bowtie \mathtt{k}| = \ldots$

$\Rightarrow$ **Cardinality estimation** must provide an estimate for each plan class.

CCPs and Join Ordering Algorithms

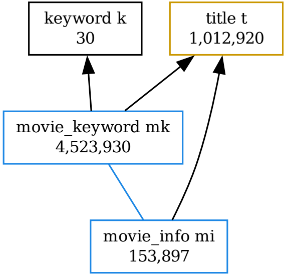


csg-cmp-pair (CCP):
two disjoint plan classes that are connected by at least one edge in the query graph.

Different alternatives to join $\text{pc}(\{\mathtt{t}, \mathtt{mk}, \mathtt{mi}\})$:

- $\text{pc}(\{\mathtt{t}\}) \bowtie \text{pc}(\{\mathtt{mk}, \mathtt{mi}\})$
- $\text{pc}(\{\mathtt{mk}\}) \bowtie \text{pc}(\{\mathtt{t}, \mathtt{mi}\})$
- $\text{pc}(\{\mathtt{mi}\}) \bowtie \text{pc}(\{\mathtt{t}, \mathtt{mk}\})$

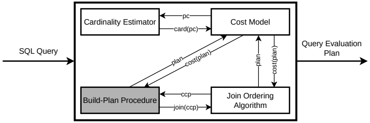
Notation e.g.: $\text{ccp}(\{\mathtt{t}\}, \{\mathtt{mk}, \mathtt{mi}\})$



Join ordering algorithms

- **DPccp**: produces cost-optimal plans, based on dynamic programming.
- **GOO**: greedy heuristics that choose the next join based on the smallest cardinality (**GooCard**) or cost (**GooCost**).

Build-Plan: Traditional Operator Selection

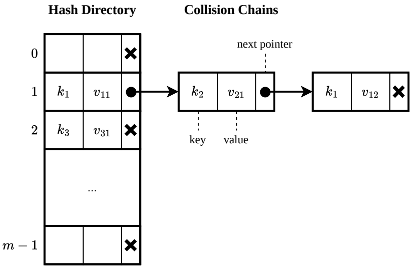


```
function BuildPlanTrad($T_1, T_2$)
    Input: two join trees $T_1, T_2$
    Output: the best join tree for joining $T_1$ and $T_2$
    $BestTree \leftarrow \mathrm{null},\ \mathrm{cost}(BestTree) \leftarrow \infty$
    for each $impl \in Implementations$ do
        $T \leftarrow T_1 \bowtie^{impl} T_2$
        if $\mathrm{cost}(BestTree) > \mathrm{cost}(T)$ then
            $BestTree \leftarrow T$
        $T \leftarrow T_2 \bowtie^{impl} T_1$
        if $\mathrm{cost}(BestTree) > \mathrm{cost}(T)$ then
            $BestTree \leftarrow T$
    return $BestTree$
```

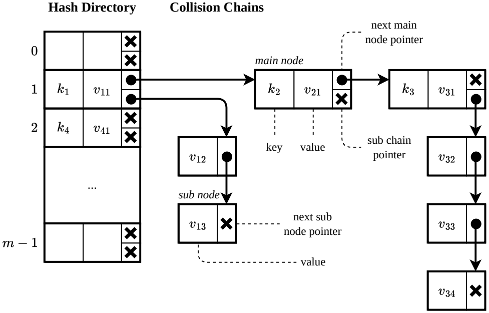
- Exhaustive search.
- Requires cost function evaluation.

Join Implementations and Alternatives

Chaining Hash Join (CH-join)



3D Hash Join (3D-join)

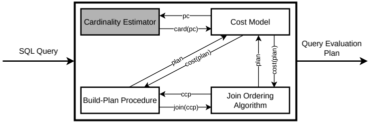


Implementation alternatives

- **Collision chain node design**: unpacked or packed collision chain nodes.
- **Prefetching**: no prefetching, rolling prefetching, AMAC [3]

[3] O. Kocberber, B. Falsafi, and B. Grot. “Asynchronous Memory Access Chaining”. In: *Proc. of the VLDB Endowment (PVLDB)* 9.4 (2015), pp. 252–263

Cardinality Estimation: $\mathrm{CE}_{\mathrm{base}}$ and $\mathrm{CE}_{\mathrm{sel}}$



Single relations (plan classes of size 1):

$$\mathrm{CE}_{\mathrm{X}}(\{R_i\}) := \begin{cases} |R_i| & \text{if X = ‘base’} \\ |\sigma_{p_i}(R_i)| & \text{if X = ‘sel’} \end{cases}$$

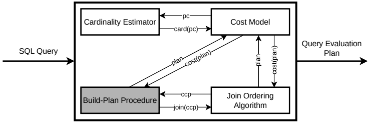
General plan classes ***S***

$$\mathrm{CE}_{\mathrm{X}}(S) := \min_{\mathrm{ccp}(S_1,S_2):\ S=S_1\cup S_2} \mathrm{CE}_{\mathrm{X}}(S,(S_1,S_2))$$

$$\mathrm{CE}_{\mathrm{X}}(S,(S_1,S_2)) := \begin{cases} \min(\mathrm{CE}_{\mathrm{X}}(S_1),\mathrm{CE}_{\mathrm{X}}(S_2)) & \text{if } \quad \mathfrak{U}(S_1) \land \quad \mathfrak{U}(S_2) \\ \mathrm{CE}_{\mathrm{X}}(S_1) & \text{if } \neg\mathfrak{U}(S_1) \land \quad \mathfrak{U}(S_2) \\ \mathrm{CE}_{\mathrm{X}}(S_2) & \text{if } \quad \mathfrak{U}(S_1) \land \neg\mathfrak{U}(S_2) \\ \mathrm{CE}_{\mathrm{X}}(S_1)\cdot\mathrm{CE}_{\mathrm{X}}(S_2) & \text{if } \neg\mathfrak{U}(S_1) \land \neg\mathfrak{U}(S_2) \end{cases}$$

$\mathfrak{U}(S_i)$: $\mathrm{ccp}(S_1, S_2)$ determines S_i uniquely, i.e., the join attributes are a (super-) key of S_i.

Build-Plan: Our Heuristic Approach



- **Goal**: Heuristic approach that does not require cost function evaluation and/or consideration of all possible alternatives.
- Two decisions to be made:
  1. Physical join operator: CH- or 3D-join.
  2. Build side of the hash join.
- **Algorithm**
  - If join attributes on both sides are unique, use the CH-join.
  - If join attributes on both sides are non-unique, use the 3D-join.
  - In both cases, choose the smaller join input as the build side.
  - If the join attributes on one side are unique, but not on the other, we make the following distinction:
    - If card(unique) $\leq 2 \cdot$ card(non-unique), then use CH-join w/ build on unique.
    - Else, use 3D-join w/ build on non-unique.

Evaluation: Overview

Dataset & Queries: Join Order Benchmark (JOB) on the IMDb dataset.

Plan generation components

- Join ordering algorithms: DPccp, GooCard, GooCost
- Build-plan procedures: $\mathrm{BP}_{\mathrm{trad}}$, $\mathbf{BP}_{\mathbf{smart}}$ (using CH- and 3D-join variants)
- Cardinality estimators: $\mathbf{CE}_{\mathbf{base}}$, $\mathbf{CE}_{\mathbf{sel}}$, $\mathrm{CE}_{\mathrm{tru}}$, $\mathrm{CE}_{\mathrm{IA\text{-}M}}$
- Cost functions: $\mathbf{CF}_{\mathbf{tru}}$, $\mathbf{CF}_{\mathbf{est}}$, $\mathrm{CF}_{\mathrm{cout}}$

Plan loss factor

Given a plan class S and a plan P for S. The *loss factor* of P is the true cost of P divided by the true cost of the overall best plan for S.

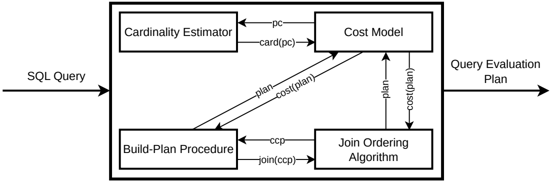
Evaluation: Results

Overall worst case across all JOB queries

- maximum loss factor: 35 785
- average loss factor: 1168

JOA	Build-Plan	Cost Fn.	Card. Est.	Plan Loss (avg)	Plan Loss (max)
DPccp	$\mathrm{BP}_{\mathrm{trad}}$	$\mathrm{CF}_{\mathrm{tru}}$	$\mathrm{CE}_{\mathrm{tru}}$	1.00	1.00
DPccp	$\mathrm{BP}_{\mathrm{smart}}$	$\mathrm{CF}_{\mathrm{cout}}$	$\mathrm{CE}_{\mathrm{base}}$	2.57	6.90
GooCard	$\mathrm{BP}_{\mathrm{smart}}$	—	$\mathrm{CE}_{\mathrm{base}}$	2.27	6.90
GooCard	$\mathrm{BP}_{\mathrm{smart}}$	—	$\mathrm{CE}_{\mathrm{sel}}$	2.32	6.71

Conclusion



In the first version of a DBMS, acceptable plan quality can be achieved using simple QO with the following building blocks:

- Heuristic **join ordering** only based on cardinality estimates: GooCard
- Very simple **cardinality estimation**: CE_{base}
- **Operator selection** using only cardinalities and uniqueness: BP_{smart}

If implemented with proper modularization, simple approaches can be replaced by more sophisticated ones step by step.