Carnegie Mellon University

OPTIMIZE!

Database Query Optimization

Background & Course Overview

COURSE OBJECTIVES

Learn about modern practices and systems programming in database query optimizers.

Students will become proficient in:
→ Query optimizer implementations
→ Writing correct + performant code
→ Proper documentation + testing

We will cover both foundational materials and state-of-the-art topics.

WHY YOU SHOULD TAKE THIS COURSE

There are more databases than ever.
Everybody has database problems.
Humans are not scalable.
Query optimization is a key system differentiator.

Research: The problem is hard / interesting.
Industry: Every database company needs this.


salesforce
salesforce
amazon
Materialize
[tile]DB
snowflake
HyPer
databricks
Google
databricks
amazon
salesforce
ROCKSET
yugabyteDB
snowflake
RisingWave
ByteDance
yugabyteDB
databricks
databricks
salesforce
impira
ORACLE
databricks
startree
snowflake
facebook
amazon
NEON
spiral
OpenAI
Google
databricks
yugabyteDB
snowflake

BACKGROUND

I assume that you have already taken an intro course on database systems (e.g., 15-445/645).

→ Things that we will **not** cover: SQL, Relational Algebra, Basic Algorithms + Data Structures, Storage Models, Query Processing

This is also **not** a ML course. We will not cover ML algorithms beyond what is discussed in papers.

→ Andy only cares about databases and whatever he can use to make databases run better.

COURSE LOGISTICS

Course Policies + Schedule:

→ Refer to course web page.

Academic Honesty:

→ Refer to CMU policy page.

→ If you're not sure, ask me.

OFFICE HOURS

After class in my office (GHC 9019):
- → Wednesdays @ 3:30 – 4:30pm
- → Or by appointment

Things that we can talk about:
- → Issues on implementing projects
- → Paper clarifications/discussion
- → How to get a database dev job.
- → DJ Mooshoo legal status

TEACHING ASSISTANTS

Head TA: Wan Shen Lim

→ 5^{th} Year PhD Student (CSD)
→ Former Paralegal
→ Certified Chicken Farmer
→ Capybara Enthusiast
→ #1 Ranked Database Ph.D. Student at Carnegie Mellon University.

GRADE BREAKDOWN

Reading Reviews (15%)
Lecture Notes (10%)
Project #1 (20%)
Project #2 (40%)
Final Exam (15%)

READING ASSIGNMENTS

One mandatory reading per class (👑).

You must submit a synopsis **before** class:

→ Overview of the main idea (three sentences).
→ Three strengths of method (one sentence each).
→ Three weaknesses of method (one sentence each).
→ Workloads evaluated (one sentence).

You are allowed to miss **one** review per semester.

You do **not** have to submit a review for book chapters.

Submission Form:
https://cmudb.io/15799-s25-submit

READING ASSIG

One mandatory reading per clas

You must submit a synopsis **bef**

→ Overview of the main idea (three s

→ Three strengths of method (one se

→ Three weaknesses of method (one

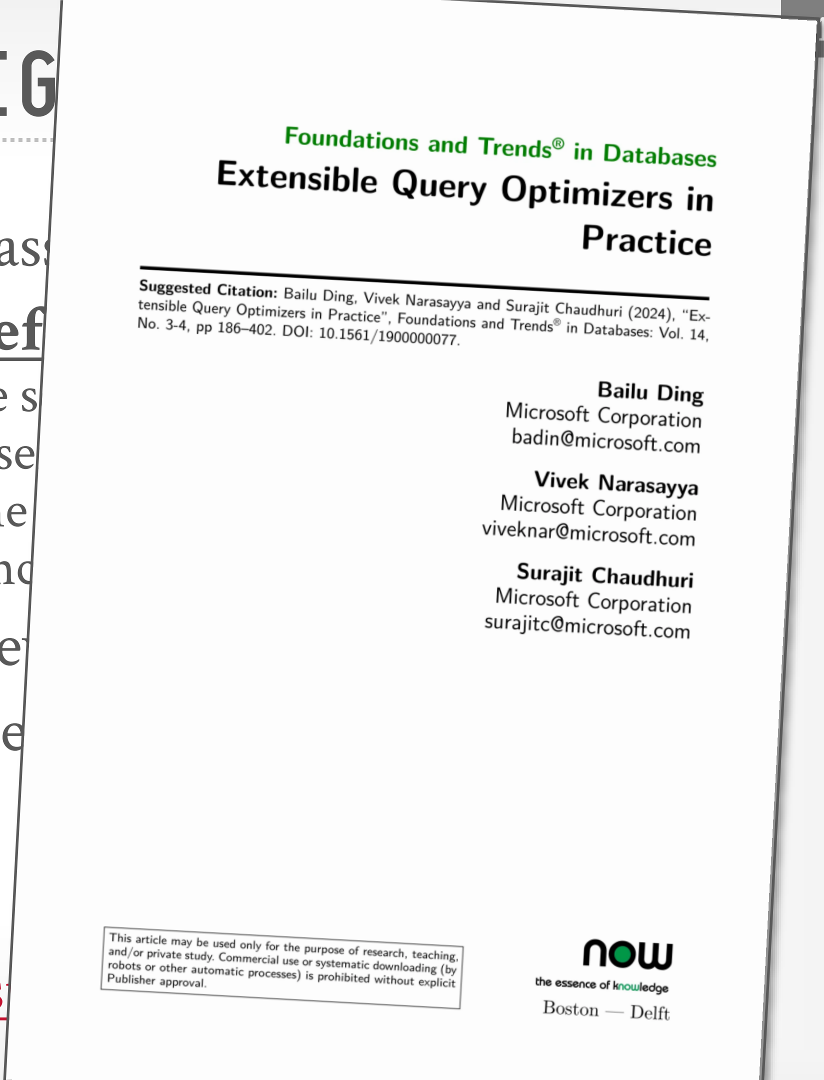
→ Workloads evaluated (one sentenc

You are allowed to miss **one** re

You do **not** have to submit a re chapters.

Submission Form:

https://cmudb.io/15799-s25-s


Foundations and Trends® in Databases

Extensible Query Optimizers in Practice

Suggested Citation: Bailu Ding, Vivek Narasayya and Surajit Chaudhuri (2024), "Extensible Query Optimizers in Practice", Foundations and Trends® in Databases: Vol. 14, No. 3-4, pp 186–402. DOI: 10.1561/1900000077.

Bailu Ding
Microsoft Corporation
badin@microsoft.com

Vivek Narasayya
Microsoft Corporation
viveknar@microsoft.com

Surajit Chaudhuri
Microsoft Corporation
surajitc@microsoft.com

This article may be used only for the purpose of research, teaching, and/or private study. Commercial use or systematic downloading (by robots or other automatic processes) is prohibited without explicit Publisher approval.

now the essence of knowledge
Boston — Delft

☠ PLAGIARISM WARNING ☠

Each review must be your own writing.

→ You may **not** copy text from the papers or other sources that you find on the web.

→ You may **not** use AI tools to generate the summary.

Plagiarism will **not** be tolerated.

See CMU's Policy on Academic Integrity for additional information.

LECTURE NOTES

Each student will be assigned one class during the semester to write notes about the lecture's contents.

→ Summarize the key topics and material from the lecture.

→ You do **not** need to include ancillary discussions from student questions or when "Andy goes off the chain".

Notes will be available on the course website + CMU-DB's Github repository.

→ Latex templates and samples are available.

→ Must include paper citations (Bibtex).

→ You are allowed to use images from course slides.

LECTURE NOTES

Each student must submit their notes as a PR within **one week** (seven days) of the lecture.

See course administration spreadsheet for your assigned lecture date.
→ You are allowed to swap without notifying instructors.

You are allowed to use AI to assist with this.
→ Please include information about what tools you used to help future students.
→ We can provide video transcripts if needed.
→ **You are responsible for the contents of the notes.**

FINAL EXAM

Written long-form examination on the readings and topics discussed in class.

Exam will be in-class on the last day of the semester.

PROJECT #1

Exploration of an existing query optimizer framework. It is purposely designed to be open ended to let you play around.

→ Anything is on the table as long as the DBMS doesn't crash, lose data, or produce incorrect query results.

We will provide you with infrastructure to run workloads.

Project #1 will be completed individually.

PROJECT #2

Each group (max 3 people) will choose a project that satisfies the following criteria:

→ Relevant to the materials discussed in class.
→ Requires a significant programming effort from all team members.
→ Unique (i.e., two groups cannot pick same idea).
→ Approved by me.

You don't have to pick a topic until after you come back from Spring Break.
We will provide sample project topics.

☠ PLAGIARISM WARNING ☠

These projects must be all your own code.

You may **not** copy source code from other groups or the web.

Plagiarism will **not** be tolerated.
See CMU's Policy on Academic Integrity for additional information.

☠ SCHEDULE WARNING ☠

For the last two years, Andy has gotten a stomach virus from the CMU Preschool the exact same week in the middle of the semester:

→ Cancelled Class: March 27th, 2024
→ Cancelled Class: April 3rd, 2023

This is likely to happen again, and we adjust the lecture schedule accordingly.

Do not acquire children before you graduate!

CMU-DB Seminar Series

Mondays @ 4:30pm (starting on Feb 3rd)

db.cs.cmu.edu/seminar2025

BEFORE WE BEGIN

There are two topics in database systems where Andy is less knowledgeable than other parts:

→ Incremental View Maintenance
→ Query Optimization

If you find an error in the lectures, please send your correction to **db-mistakes@cs.cmu.edu**.

Query Optimization

QUERY OPTIMIZATION

Query languages like SQL are **declarative**.
→ The user tells the DBMS what result they want and (usually) not how to compute it.

```
SELECT DISTINCT ename
  FROM Emp E JOIN Dept D
    ON E.did = D.did
 WHERE D.dname = 'Toy'
```

For a given query, the DBMS attempts to find a **correct** execution plan with the best **cost**.

This is what this course is about!

MOTIVATION

```
SELECT DISTINCT ename
  FROM Emp E JOIN Dept D
    ON E.did = D.did
 WHERE D.dname = 'Toy'
```

Catalog

clustered *unclustered* *unclustered*

Emp(<u>ssn</u>,ename,addr,sal,did)

10,000 records
1,000 pages

clustered *unclustered*

Dept(<u>did</u>,dname,floor,mgr)

500 records
50 pages

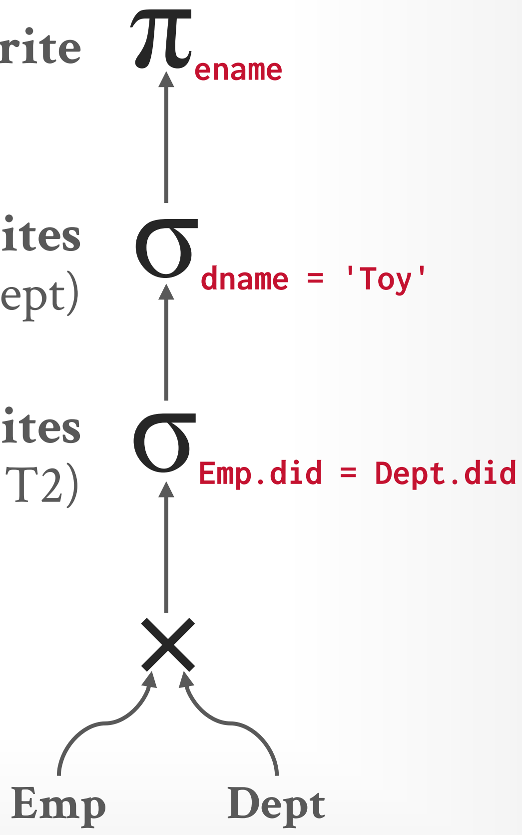
Total: 2M I/Os

4 reads + 1 write

2,000 reads + 4 writes
(10K/500 = 20 emps per dept)

1,000,000 reads + 2,000 writes
(FK join, 10k tuples in temp T2)

(50 + 50,000) reads
+ 1,000,000 writes
Write temp file T1
5 tuples per page in T1



MOTIVATION

```sql
SELECT DISTINCT ename
  FROM Emp E JOIN Dept D
    ON E.did = D.did
 WHERE D.dname = 'Toy'
```

Catalog

clustered *unclustered* *unclustered*

Emp(ssn,ename,addr,sal,did)

10,000 records
1,000 pages

clustered *unclustered*

Dept(did,dname,floor,mgr)

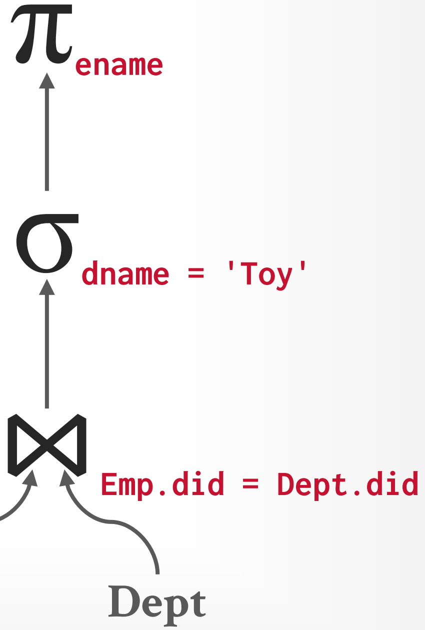
500 records
50 pages

Total: 54k I/Os

4 reads + 4 writes
Read temp T2

2,000 reads + 4 writes
Read temp T1, Write temp T2

(50 + 50,000) reads
+ 2,000 writes
Page Nested-Loop Join
Write Temp T1



MOTIVATION

```sql
SELECT DISTINCT ename
  FROM Emp E JOIN Dept D
    ON E.did = D.did
 WHERE D.dname = 'Toy'
```

Catalog

clustered *unclustered* *unclustered*

Emp(<u>ssn</u>,ename,addr,sal,did)

10,000 records
1,000 pages

clustered *unclustered*

Dept(<u>did</u>,dname,floor,mgr)

500 records
50 pages

Total: 54k I/Os

4 reads + 4 writes
Read temp T2

π_{ename}

2,000 reads + 4 writes
Read temp T1, Write temp T2

$\sigma_{dname = 'Toy'}$

(50 + 50,000) reads
+ 2,000 writes
Page Nested-Loop Join
Write Temp T1

$\bowtie_{Emp.did = Dept.did}$

Emp **Dept**

MOTIVATION

```
SELECT DISTINCT ename
  FROM Emp E JOIN Dept D
    ON E.did = D.did
 WHERE D.dname = 'Toy'
```

Catalog

clustered *unclustered* *unclustered*

Emp(ssn,ename,addr,sal,did)

10,000 records
1,000 pages

clustered *unclustered*

Dept(did,dname,floor,mgr)

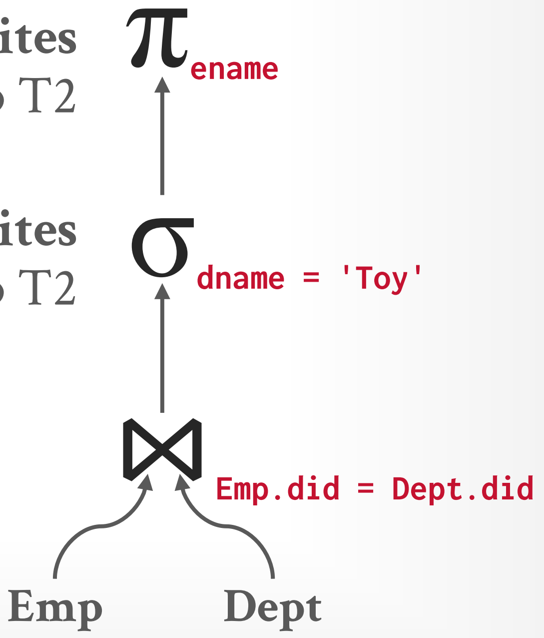
500 records
50 pages

Materialization Model → **Total: 7,159 I/Os**

4 reads + 4 writes
Read temp T2

2,000 reads + 4 writes
Read temp T1, Write temp T2

3×(|Emp| + |Dept|) =
3,150 reads + 2,000 writes
Sort-Merge Join (50 Buffers)
Write Temp T1



MOTIVATION

```sql
SELECT DISTINCT ename
  FROM Emp E JOIN Dept D
   ON E.did = D.did
 WHERE D.dname = 'Toy'
```

Catalog

clustered *unclustered* *unclustered*

Emp(ssn,ename,addr,sal,did)

10,000 records
1,000 pages

clustered *unclustered*

Dept(did,dname,floor,mgr)

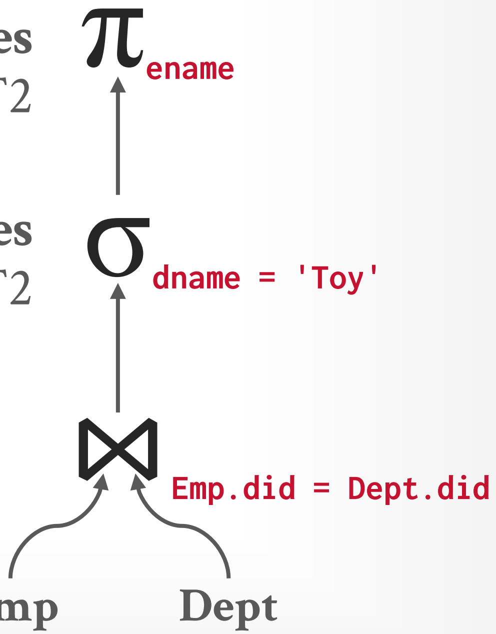
500 records
50 pages

No Pipelining!
↳ **Materialization Model** → **Total: 7,159 I/Os**

4 reads + 4 writes
Read temp T2

2,000 reads + 4 writes
Read temp T1, Write temp T2

3×(|Emp| + |Dept|) =
3,150 reads + 2,000 writes
Sort-Merge Join (50 Buffers)
Write Temp T1



MOTIVATION

```
SELECT DISTINCT ename
  FROM Emp E JOIN Dept D
    ON E.did = D.did
 WHERE D.dname = 'Toy'
```

Vectorization Model → Total: 3,151 I/Os

No Pipelining! ⮑ Materialization Model → Total: 7,159 I/Os

Catalog

clustered ▲ *unclustered* △ *unclustered* △

Emp(ssn,ename,addr,sal,did)

10,000 records
1,000 pages

clustered ▲ *unclustered* △

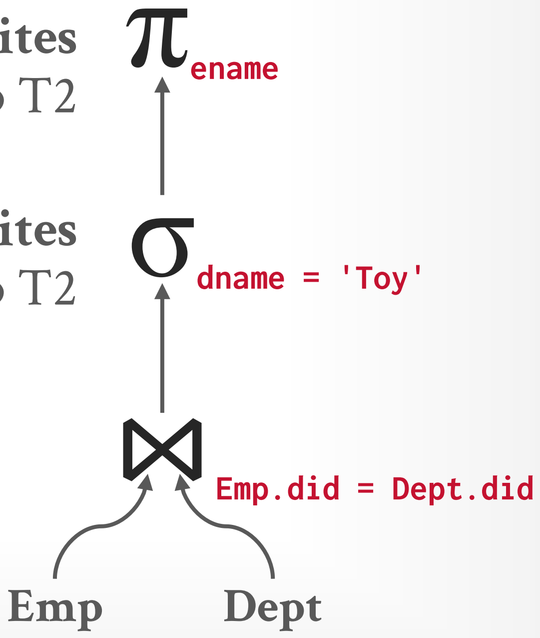
Dept(did,dname,floor,mgr)

500 records
50 pages

reads + 4 writes
Read temp T2

2 00 reads + writes
Read temp T1, Write temp T2

3×(|Emp| + |Dept|) =
3,150 reads + 2, 0 writes
Sort-Merge Join (50 Buffers)
Write Temp T1



MOTIVATION

```
SELECT DISTINCT ename
  FROM Emp E JOIN Dept D
    ON E.did = D.did
 WHERE D.dname = 'Toy'
```

Catalog

clustered (ssn) *unclustered* (ename) *unclustered* (did)

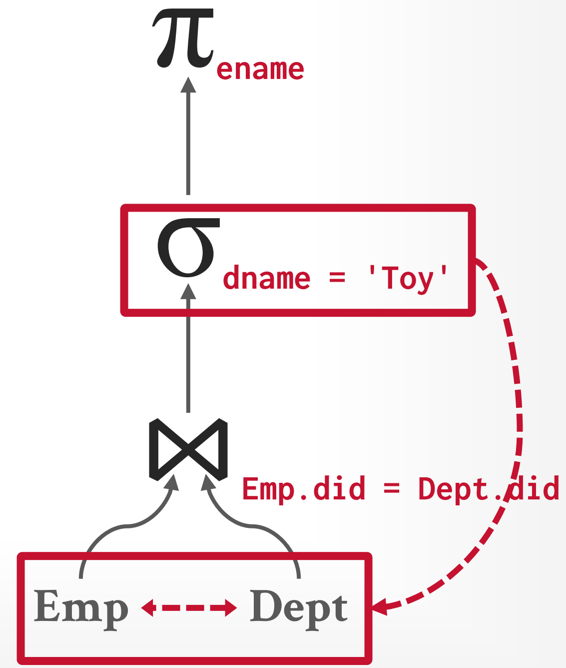
Emp(<u>ssn</u>,ename,addr,sal,did)

10,000 records
1,000 pages

clustered (did) *unclustered* (dname)

Dept(<u>did</u>,dname,floor,mgr)

500 records
50 pages



MOTIVATION

```
SELECT DISTINCT ename
  FROM Emp E JOIN Dept D
    ON E.did = D.did
 WHERE D.dname = 'Toy'
```

Catalog

clustered (ssn), *unclustered* (ename), *unclustered* (did)

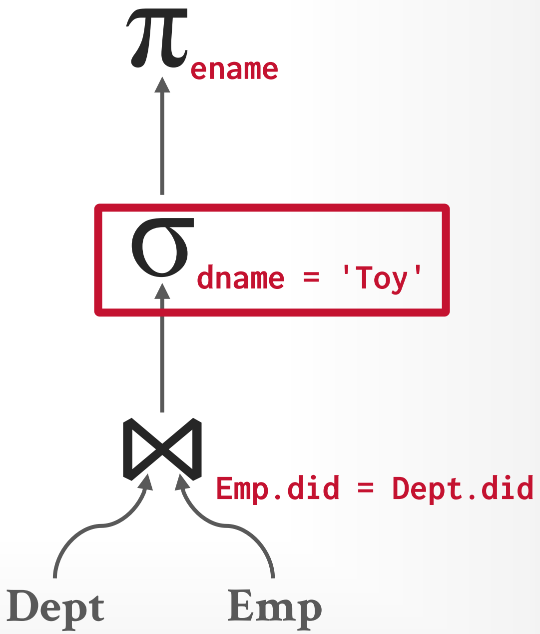
Emp(ssn,ename,addr,sal,did)

10,000 records
1,000 pages

clustered (did), *unclustered* (dname)

Dept(did,dname,floor,mgr)

500 records
50 pages



MOTIVATION

```
SELECT DISTINCT ename
  FROM Emp E JOIN Dept D
    ON E.did = D.did
 WHERE D.dname = 'Toy'
```

Catalog

clustered (ssn) *unclustered* (ename) *unclustered* (did)

Emp(<u>ssn</u>,ename,addr,sal,did)

10,000 records
1,000 pages

clustered (did) *unclustered* (dname)

Dept(<u>did</u>,dname,floor,mgr)

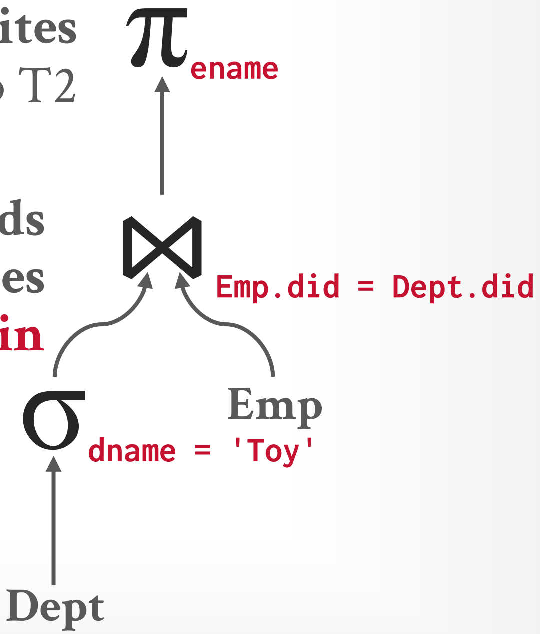
500 records
50 pages

Total: 37 I/Os

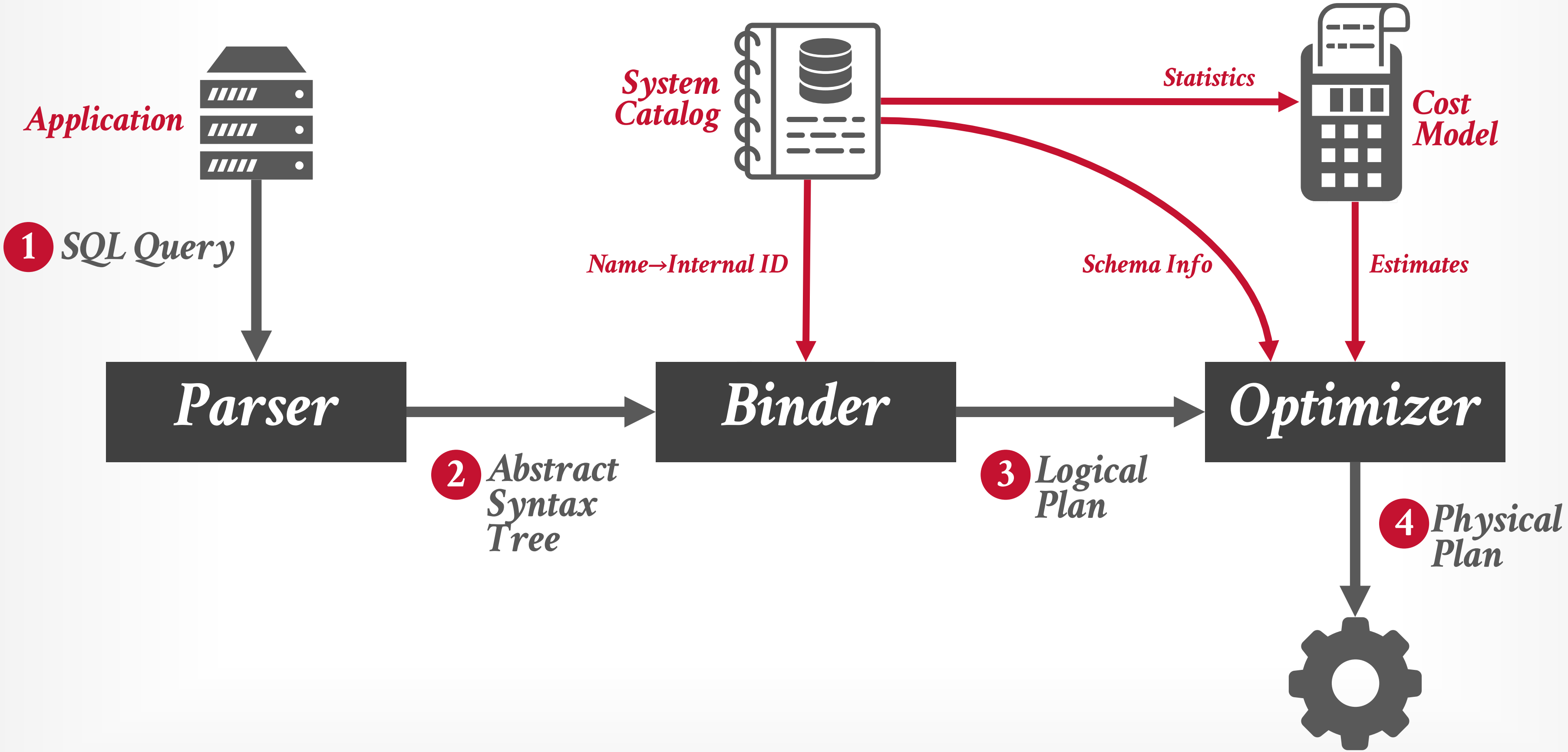
4 reads + 1 writes
Read temp T2

1 + 3 (idx) + 20 (ptr chase) reads + 4 writes
Index Nested-Loop Join

3 reads + 1 writes
Access: Index(dname)



DBMS OVERVIEW



QUERY OPTIMIZER

Takes in a **logical query plan** and generates a **physical execution plan**. The goal of this component is to:

→ Consider a large search space of promising plans
→ Accurately distinguish whether one potential plan is better than another.
→ Efficiently search the solution space to find a physical plan with the lowest cost.

Ideally an optimizer should always generate the best plan regardless of how the query is expressed.

EXTENSIBLE QUERY OPTIMIZERS IN PRACTICE
FOUNDATIONS AND TRENDS IN DATABASES 2024

LOGICAL VS. PHYSICAL PLANS

The optimizer generates a mapping of a **logical** algebra expression to the optimal equivalent physical algebra expression.

Physical operators define a specific execution strategy using an access path.

→ They can depend on the physical format of the data that they process (i.e., sorting, compression).

→ Not always a 1:1 mapping from logical to physical.

COURSE TOPICS

Search Strategies
Enumeration / Transformations
Parallelization
Statistics / Summarization
Cardinality Estimation / Parameterization
Adaptivity / Feedback Mechanisms
Real-world Implementations

SEARCH STRATEGIES

Heuristics / Rules

→ Rewrite the query to remove (guessed) inefficiencies.
→ Examples: always do selections first or push down projections as early as possible.
→ These techniques may need to examine catalog, but they do not need to examine data.

Cost-based Search

→ Use a model to estimate the cost of executing a plan.
→ Enumerate multiple equivalent plans for a query and pick the one with the lowest cost.

TOP-DOWN VS. BOTTOM-UP

Bottom-up Optimization

→ Start with nothing and then build up the plan to get to the outcome that you want.

→ **Examples**: System R, Starburst

Top-down Optimization

→ Start with the outcome that the query wants and then work down the tree to find the optimal plan that gets you to that goal.

→ **Examples**: Volcano, Cascades

OPTIMIZATION OBJECTIVE

There are several goals a DBMS can consider when optimizing a query:

→ Minimize Response Time
→ Minimize Resource Consumption
→ Minimize Monetary Cost
→ Maximize Throughput

The DBMS uses a **cost model** to predict the behavior of a query plan given a database state.

→ This is an internal cost that allows the DBMS to compare one plan with another.

OPTIMIZATION GRANULARITY

Choice #1: Single Query

→ Much smaller search space.
→ DBMS (usually) does not reuse results across queries.
→ To account for resource contention, the cost model must consider what is currently running.

Choice #2: Multiple Queries

→ More efficient if there are many similar queries.
→ Search space is much larger.
→ Useful for data / intermediate result sharing.

PARTING THOUGHTS

This is very hard.

This is the part of a DBMS that is the hardest to implement well (proven to be NP-Complete).

→ Queries will have multiple alternative plans with different runtime characteristics.

No optimizer truly produces the "optimal" plan.

→ Use estimation techniques to guess real plan cost.

→ Use heuristics to limit the search space.

NEXT CLASS

IBM System R Optimizer
→ *The OG Implementation*

Make sure that you submit the first reading review

https://cmudb.io/15799-s25-submit