Carnegie Mellon University

14

Query Planning & Optimization – Part I

Intro to Database Systems
15-445/15-645
Fall 2019

Andy Pavlo
Computer Science
Carnegie Mellon University

ADMINISTRIVIA

Mid-Term Exam is Wed Oct 16th @ 12:00pm
→ See mid-term exam guide for more info.

Project #2 is due Sun Oct 20th @ 11:59pm

QUERY OPTIMIZATION

Remember that SQL is declarative.

- → User tells the DBMS what answer they want, not how to get the answer.

There can be a big difference in performance based on plan is used:

- → See last week: 1.3 hours vs. 0.45 seconds

IBM SYSTEM R

First implementation of a query optimizer from the 1970s.

- People argued that the DBMS could never choose a query plan better than what a human could write.

Many concepts and design decisions from the **System R** optimizer are still used today.

QUERY OPTIMIZATION

Heuristics / Rules

→ Rewrite the query to remove stupid / inefficient things.

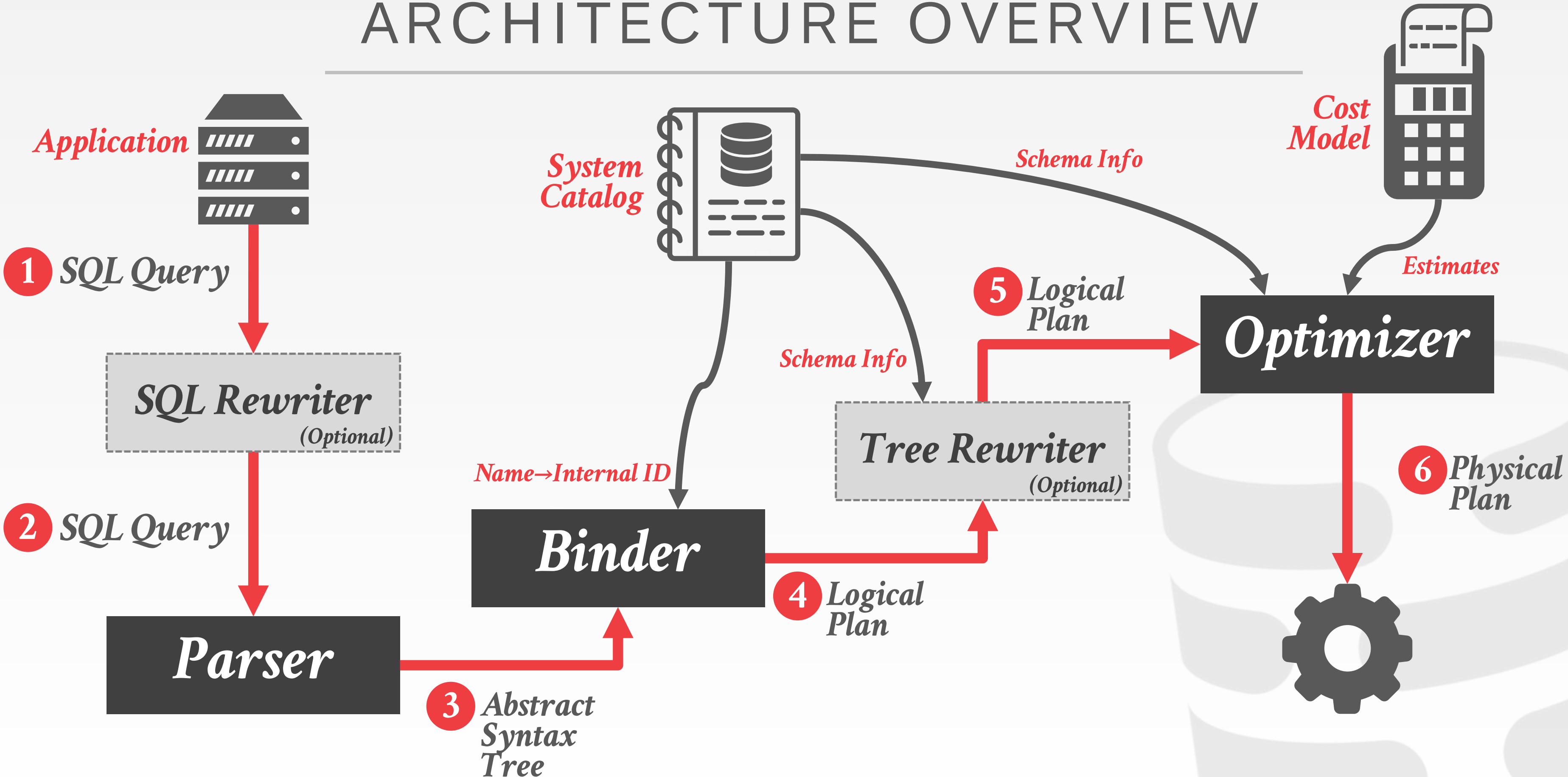
→ These techniques may need to examine catalog, but they do not need to examine data.

Cost-based Search

→ Use a model to estimate the cost of executing a plan.

→ Evaluate multiple equivalent plans for a query and pick the one with the lowest cost.

ARCHITECTURE OVERVIEW



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.

QUERY OPTIMIZATION IS NP-HARD

This is the hardest part of building a DBMS.

If you are good at this, you will get paid $$$.

People are starting to look at employing ML to improve the accuracy and efficacy of optimizers.

I am expanding the Advanced DB Systems class to cover this topic in greater detail.

TODAY'S AGENDA

Relational Algebra Equivalences

Static Rules

RELATIONAL ALGEBRA EQUIVALENCES

Two relational algebra expressions are equivalent if they generate the same set of tuples.

The DBMS can identify better query plans without a cost model.

This is often called **query rewriting**.

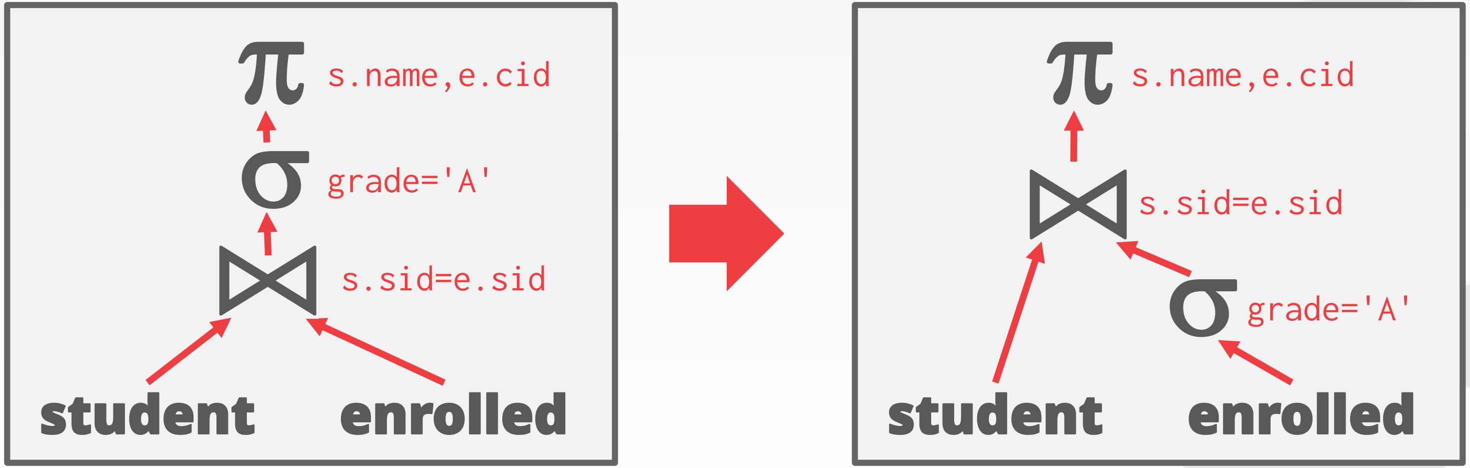
PREDICATE PUSHDOWN

```
SELECT s.name, e.cid
  FROM student AS s, enrolled AS e
 WHERE s.sid = e.sid
   AND e.grade = 'A'
```

$$\pi_{\text{name, cid}}(\sigma_{\text{grade='A'}}(\text{student} \bowtie \text{enrolled}))$$

PREDICATE PUSHDOWN

```
SELECT s.name, e.cid
  FROM student AS s, enrolled AS e
 WHERE s.sid = e.sid
   AND e.grade = 'A'
```



RELATIONAL ALGEBRA EQUIVALENCES

```
SELECT s.name, e.cid
  FROM student AS s, enrolled AS e
 WHERE s.sid = e.sid
   AND e.grade = 'A'
```

$$\pi_{\text{name, cid}}(\sigma_{\text{grade='A'}}(\text{student} \bowtie \text{enrolled}))$$

$$=$$

$$\pi_{\text{name, cid}}(\text{student} \bowtie (\sigma_{\text{grade='A'}}(\text{enrolled})))$$

RELATIONAL ALGEBRA EQUIVALENCES

Selections:

- → Perform filters as early as possible.
- → Reorder predicates so that the DBMS applies the most selective one first.
- → Break a complex predicate, and push down

$$\sigma_{p1\wedge p2\wedge \ldots pn}(\mathbf{R}) = \sigma_{p1}(\sigma_{p2}(\ldots\sigma_{pn}(\mathbf{R})))$$

Simplify a complex predicate

- → `(X=Y AND Y=3)` → `X=3 AND Y=3`

RELATIONAL ALGEBRA EQUIVALENCES

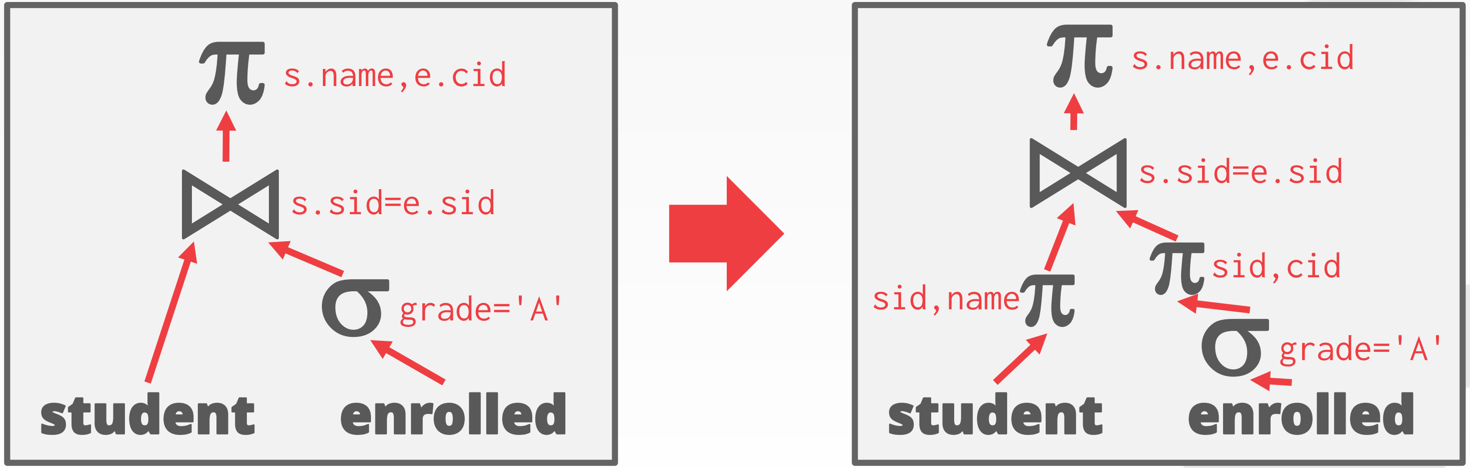
Projections:

- → Perform them early to create smaller tuples and reduce intermediate results (if duplicates are eliminated)
- → Project out all attributes except the ones requested or required (e.g., joining keys)

This is not important for a column store…

PROJECTION PUSHDOWN

```
SELECT s.name, e.cid
  FROM student AS s, enrolled AS e
 WHERE s.sid = e.sid
   AND e.grade = 'A'
```



```
CREATE TABLE A (
  id INT PRIMARY KEY,
  val INT NOT NULL );
```

MORE EXAMPLES

Impossible / Unnecessary Predicates

```
SELECT * FROM A WHERE 1 = 0;
```

✗

Source: Lukas Eder

```
CREATE TABLE A (
  id INT PRIMARY KEY,
  val INT NOT NULL );
```

MORE EXAMPLES

Impossible / Unnecessary Predicates

```
SELECT * FROM A WHERE 1 = 0;
```

✗

```
SELECT * FROM A WHERE 1 = 1;
```

Source: Lukas Eder

```
CREATE TABLE A (
  id INT PRIMARY KEY,
  val INT NOT NULL );
```

MORE EXAMPLES

Impossible / Unnecessary Predicates

```
SELECT * FROM A WHERE 1 = 0; X
```

```
SELECT * FROM A;
```

Source: Lukas Eder

```
CREATE TABLE A (
  id INT PRIMARY KEY,
  val INT NOT NULL );
```

MORE EXAMPLES

Impossible / Unnecessary Predicates

```
SELECT * FROM A WHERE 1 = 0; ✗
```

```
SELECT * FROM A;
```

Join Elimination

```
SELECT A1.*
  FROM A AS A1 JOIN A AS A2
    ON A1.id = A2.id;
```

Source: Lukas Eder

```
CREATE TABLE A (
  id INT PRIMARY KEY,
  val INT NOT NULL );
```

MORE EXAMPLES

Impossible / Unnecessary Predicates

```
SELECT * FROM A WHERE 1 = 0;
```

✗

```
SELECT * FROM A;
```

Join Elimination

```
SELECT * FROM A;
```

Source: Lukas Eder

```
CREATE TABLE A (
  id INT PRIMARY KEY,
  val INT NOT NULL );
```

MORE EXAMPLES

Ignoring Projections

```
SELECT * FROM A AS A1
 WHERE EXISTS(SELECT val FROM A AS A2
                WHERE A1.id = A2.id);
```

Source: Lukas Eder

```
CREATE TABLE A (
  id INT PRIMARY KEY,
  val INT NOT NULL );
```

MORE EXAMPLES

Ignoring Projections

```
SELECT * FROM A;
```

Source: Lukas Eder

```
CREATE TABLE A (
  id INT PRIMARY KEY,
  val INT NOT NULL );
```

MORE EXAMPLES

Ignoring Projections

```
SELECT * FROM A;
```

Merging Predicates

```
SELECT * FROM A
 WHERE val BETWEEN 1 AND 100
    OR val BETWEEN 50 AND 150;
```

Source: Lukas Eder

```
CREATE TABLE A (
  id INT PRIMARY KEY,
  val INT NOT NULL );
```

MORE EXAMPLES

Ignoring Projections

```
SELECT * FROM A;
```

Merging Predicates

```
SELECT * FROM A
 WHERE val BETWEEN 1 AND 100
    OR val BETWEEN 50 AND 150;
```

Source: Lukas Eder

```
CREATE TABLE A (
  id INT PRIMARY KEY,
  val INT NOT NULL );
```

MORE EXAMPLES

Ignoring Projections

```
SELECT * FROM A;
```

Merging Predicates

```
SELECT * FROM A
 WHERE val BETWEEN 1 AND 150;
```

Source: Lukas Eder

RELATIONAL ALGEBRA EQUIVALENCES

Joins:

→ Commutative, associative

$R \bowtie S = S \bowtie R$

$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$

How many different orderings are there for an *n*-way join?

RELATIONAL ALGEBRA EQUIVALENCES

How many different orderings are there for an *n*-way join?

Catalan number **≈4^n**

→ Exhaustive enumeration will be too slow.

We'll see in a second how an optimizer limits the search space...

CONCLUSION

We can use static rules and heuristics to optimize a query plan without needing to understand the contents of the database.

NEXT CLASS

MID-TERM EXAM!

- → Seriously, this is not a joke.