Lecture 19: Rule-Based Query Optimization

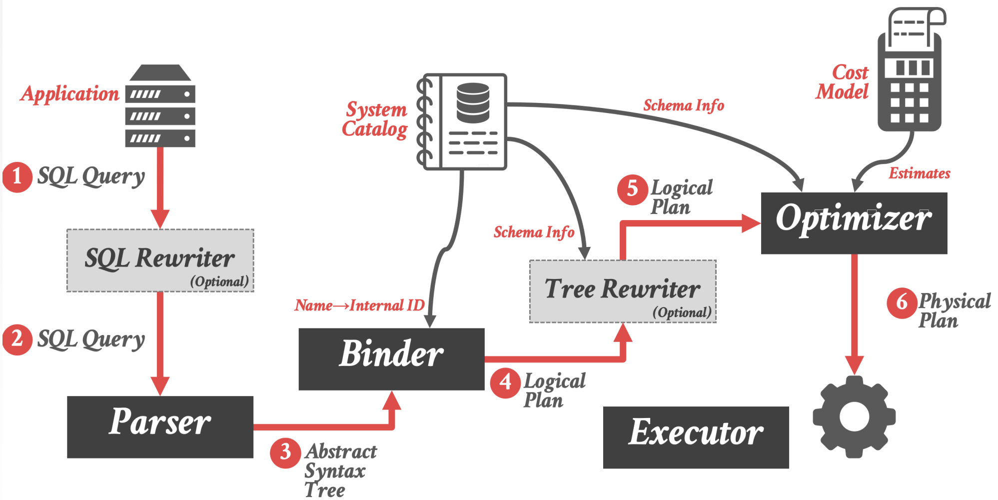
Today's Agenda

Administrivia

- Assignment 3 and Sheet 3 due on April 5 (Tue)
- Project checkpoint on April 12 (Tue)

Recap

Anatomy of a Database System [Monologue]



Anatomy of a Database System [Monologue]

- Process Manager
  - Manages client connections
- Query Processor
  - Parse, plan and execute queries on top of storage manager
- Transactional Storage Manager
  - Knits together buffer management, concurrency control, logging and recovery
- Shared Utilities
  - Manage hardware resources across threads

Anatomy of a Database System [Monologue]

- Process Manager
  - Connection Manager + Admission Control
- Query Processor
  - Query Parser
  - Query Optimizer (*a.k.a.*, Query Planner)
  - Query Executor
- Transactional Storage Manager
  - Lock Manager
  - Access Methods (*a.k.a.*, Indexes)
  - Buffer Pool Manager
  - Log Manager
- Shared Utilities
  - Memory, Disk, and Networking Manager

Motivation

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:
  - Hours $\rightarrow$ Seconds

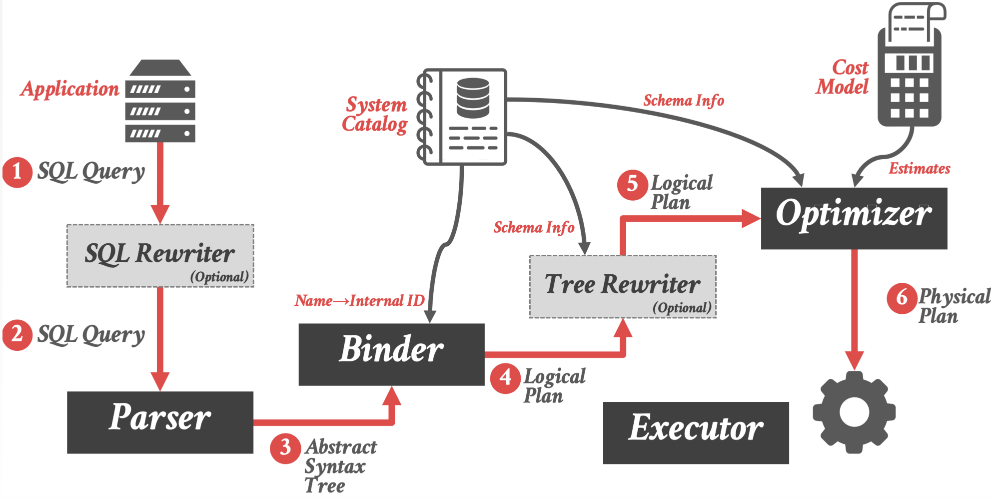
IBM System R

- 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

- **Approach 1: 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.
- **Approach 2: 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.

Anatomy of a Database System [Monologue]



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

Relational Algebra Equivalences

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**.

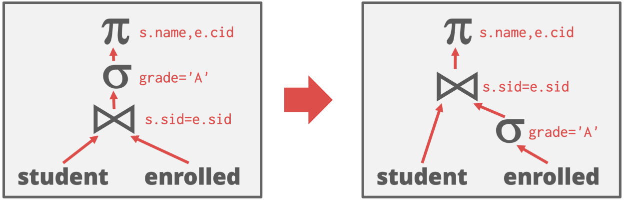
Predicate Pushdown

```
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

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



Predicate Pushdown

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

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

$$=$$

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

Relational Algebra Equivalences

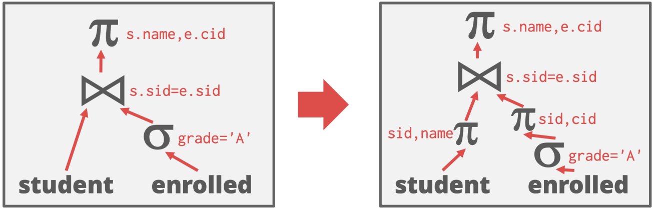
- Selections:
  - Perform filters as early as possible.
  - Reorder predicates so that the DBMS applies the most selective one first.
  - Break a complex predicate, and push down $\sigma_{p_1 \wedge p_2 \wedge \ldots p_n}(R) = \sigma_{p_1}(\sigma_{p_2}(\ldots \sigma_{p_n}(R)))$
- Simplify a complex predicate
  - (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

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



Impossible / Unnecessary Predicates

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

```
SELECT * FROM A WHERE 1 = 0; --- No Results
SELECT * FROM A WHERE 1 = 1;
```

```
SELECT * FROM A;
```

Join Elimination

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

```
SELECT * FROM A;
```

Ignoring Projections

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

```
 SELECT * FROM A;
```

Georgia Tech

Merging Predicates

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

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

Relational Algebra Equivalences

- Joins:
  - Commutative: $R \bowtie S = S \bowtie R$
  - Associative: $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$
- 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: $\approx 4_n$
  - Exhaustive enumeration will be too slow.
  - We'll later look at how an optimizer **prunes** the **search space**.

Nested Sub-Queries

Nested Sub-Queries

- The DBMS treats nested sub-queries in the *WHERE* clause as functions that take parameters and return a single value or set of values.
- **Two Approaches**:
  - Rewrite to **de-correlate** and/or flatten them
  - Decompose nested query and store result to **temporary table**

Nested Sub-Queries: Rewrite

```
SELECT name FROM sailors AS S
 WHERE EXISTS (
    SELECT * FROM reserves AS R
     WHERE S.sid = R.sid
       AND R.day = '2018-10-15'
 )
```

```
SELECT name
  FROM sailors AS S, reserves AS R
 WHERE S.sid = R.sid
   AND R.day = '2018-10-15'
```

Nested Sub-Queries: Decompose

- For each sailor with the highest rating (over all sailors) and at least two reservations for red boats, find the sailor id and the earliest date on which the sailor has a reservation for a red boat.

```
SELECT S.sid, MIN(R.day)
  FROM sailors S, reserves R, boats B
 WHERE S.sid = R.sid
   AND R.bid = B.bid
   AND B.color = 'red'
   AND S.rating = (SELECT MAX(S2.rating)
                     FROM sailors S2)
 GROUP BY S.sid
HAVING COUNT(*) > 1
```

Decomposing Queries

- For harder queries, the optimizer breaks up queries into blocks and then concentrates on one block at a time.
- Sub-queries are written to a temporary table that are discarded after the query finishes.

Decomposing Queries

```sql
SELECT S.sid, MIN(R.day) --- Outer Block
  FROM sailors S, reserves R, boats B
 WHERE S.sid = R.sid
   AND R.bid = B.bid
   AND B.color = 'red'
   AND S.rating = () --- Result of Nested Query
 GROUP BY S.sid
HAVING COUNT(*) > 1
```

```sql
SELECT MAX(rating) FROM sailors
```

Conclusion

Query Optimization

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

Next Class

- Cost-based Query Optimization