Discussion 7 Query Optimization

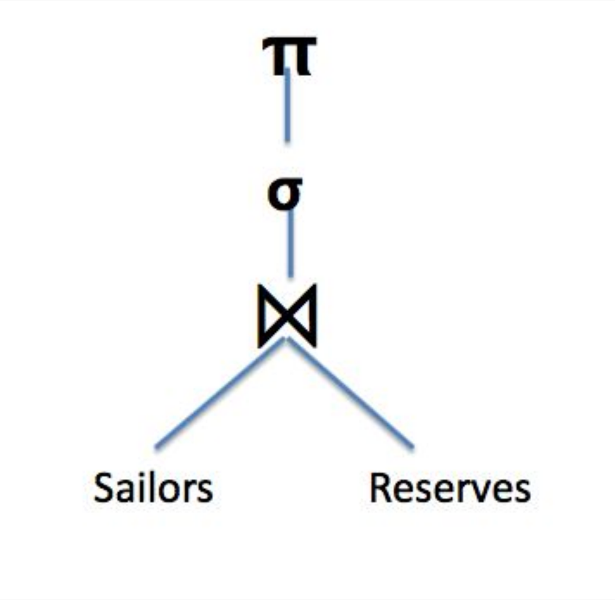
Weijie Lyu

Reminder

1. Congratulations on finished Midterm!
2. Homework 2 due on Nov. 13, 23:59

Query Optimization - Background

- We can represent relational algebra expressions as trees
- Order of operators affects I/Os and resource usage, but not necessarily output



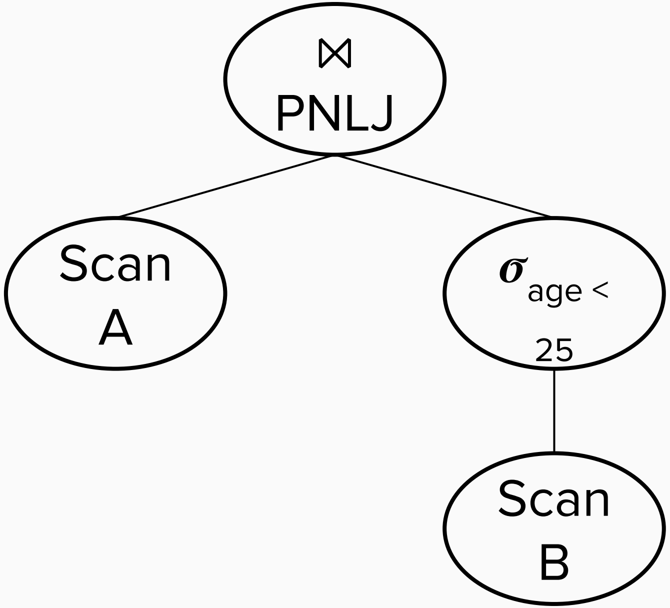
Query Optimization - Alternate Plans

- Given a plan, some things we can do are:
  - **Push selections/projections down the tree**
    - The earlier we reduce the size of our input data, the fewer I/Os are incurred as we traverse up the tree
    - Only affects I/O cost if materialized, or if operator only makes one pass (so not right relation of BNLJ)

Query Optimization - Alternate Plans

- Given a plan, some things we can do are:
  - **Push selections/projections down the tree**
  - **Materialize intermediate relations** (write to a temp file)
    - Results in additional write I/Os, but is better in the long run
  - **Use indices** (e.g. INLJ)

Query Optimization - Materializing



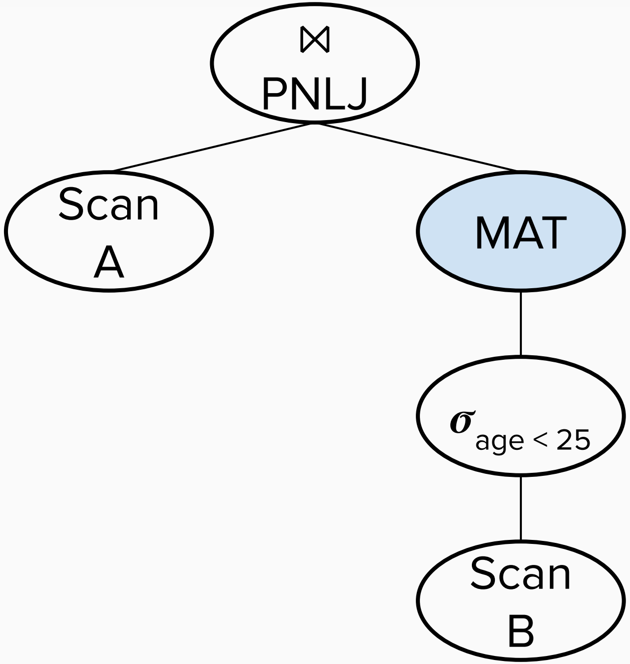
- Table A: takes 50 I/Os to perform a scan
- Table B: takes 100 I/Os to perform a scan
- Sel(B.age < 25) = 0.5, [B] = 100

Without materializing, we're performing $\sigma_{age < 25}$ on the fly each time in PNLJ, and scanning the entire table B for each page of A.

Cost = *Scan A* (50) + *PNLJ* (50*100)

→ 5,050 I/Os in total

Query Optimization - Materializing



- Table A: takes 50 I/Os to perform a scan
- Table B: takes 100 I/Os to perform a scan
- Sel(B.age < 25) = 0.5, [B] = 100

By materializing the intermediate relation, we're applying $\sigma_{age < 25}$ before PNLJ, and performing the join on the *result* of the selection.

Cost = *Scan A* (50) + *Scan B* (100)

+ *Materialize* (100 * 0.5) + *PNLJ* (50 * 50)

➜ 2,700 I/Os in total

Query Optimization

- A query optimizer takes in a query plan (e.g. one directly translated from a SQL query), and outputs a better (hopefully optimal) query plan
    - Works on and optimizes over a **plan space** (set of all plans considered)
    - Performs **cost estimation** on query plans
    - Uses a **search algorithm** to search through plan space to find plan with lowest cost estimate
        - May not be optimal (bad estimate, or small plan space)

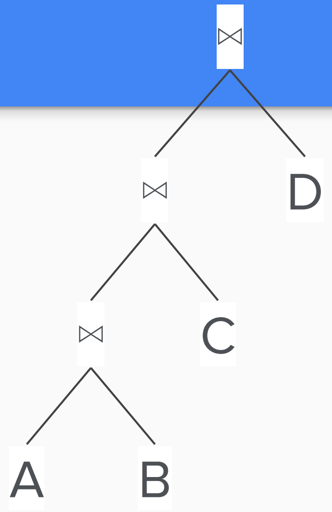
Query Optimization - Selinger

- We'll be looking at the System R optimizer (aka Selinger optimizer)
  - **Plan space:** only **left-deep** trees, avoid cartesian products unless they're the only option.
    - **Left-deep trees** represent a plan where all new tables are joined one at a time from the right.
  - **Cost estimation:** actual Selinger optimizer incorporates both CPU and I/O cost; we'll only use I/O cost for this class
  - **Search algorithm:** dynamic programming

Query Optimization - Selinger

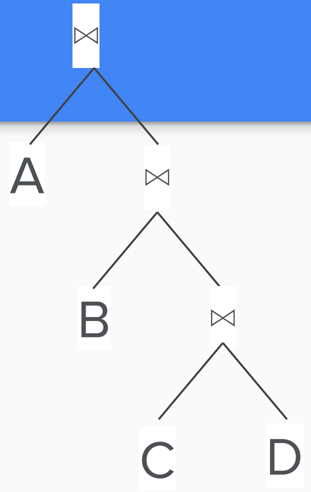
- Why only left-deep trees?
  - Join new tables one at a time from the right
  - Create an ordering in which to add tables to the query being executed
  - Too many possible trees for joins
    - Using only left-deep trees: N! different ways to order relations
    - Including all permutations tree layouts: A very large number of ways to parenthesize given an ordering (superexponential in N)

Query Optimization - Selinger



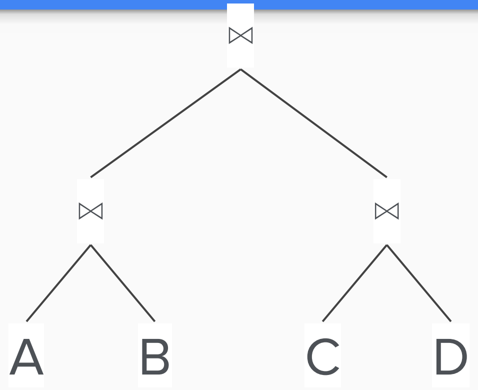
Left-deep

$((A \bowtie B) \bowtie C) \bowtie D$



Right-deep

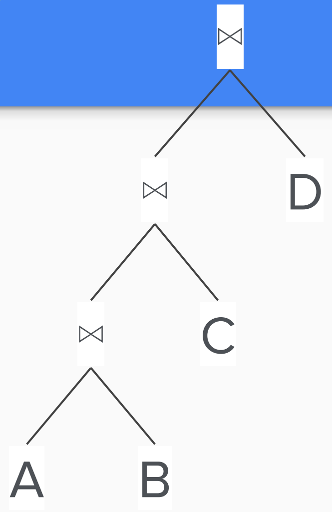
$A \bowtie (B \bowtie (C \bowtie D))$



Bushy

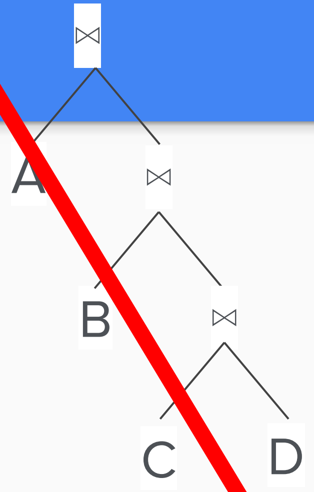
$(A \bowtie B) \bowtie (C \bowtie D)$

Only consider left-deep plans!



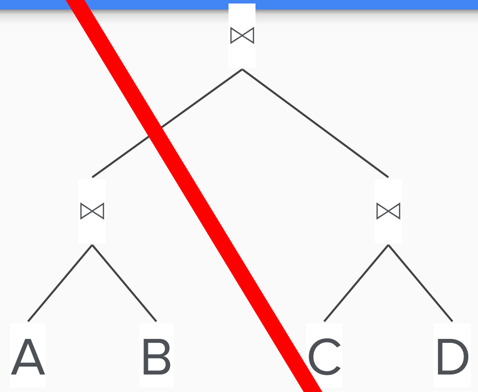
Left-deep

$((A \bowtie B) \bowtie C) \bowtie D$



Right-deep

$A \bowtie (B \bowtie (C \bowtie D))$



Bushy

$(A \bowtie B) \bowtie (C \bowtie D)$

Query Optimization - Selinger

- Search algorithm for Selinger: use dynamic programming
  - Runtime drops from **n!** to around **$n*2^n$**
- To be considered, must be:
  - Left deep
  - No cartesian products (i.e. if we join R and S on <cond1> and we join S and T on <cond2>, we don't consider joining R and T if there's no condition)

Query Optimization - Selinger

- For n relations joined, perform n passes
  - on the i-th path, output only the best plan for joining any i of the n relations
  - Also keep around plans that have higher cost but have an **interesting order**

Query Optimization - Interesting Orders

- **Interesting orders** are orderings on intermediate relations that ***may*** help reduce the cost of later joins
  - ORDER BY attributes
  - GROUP BY attributes
  - *downstream* join attributes

Query Optimization - Selinger

- **Pass 1:** find minimum cost access method for each (relation, interesting order) pair
  - Index scan, full table scans

A toy example:

SELECT *

FROM A, B, C

Pass 1:

- Full scan on A: 2 I/Os
- Index scan on A.b: 1 I/Os
- Full scan on B: 2 I/Os
- Full scan on C: 4 I/Os
- Index scan on C.c: 2 I/Os
- Index scan on C.d: 3 I/Os

Query Optimization - Selinger

- **Pass i** (Repeat until all relations are joined)**:**
  take in list of optimal plans for (i - 1 relations, interesting order) from Pass i-1, and compute minimum cost plan for (i relations, interesting orders) (every size i subset of the n relations)

Pass 2:

- Index scan on A.b: 1 I/Os
- Full scan on B: 2 I/Os
- Index scan on C.c: 2 I/Os

- A BNLJ B: 5 I/Os
- B INLJ A: 6 I/Os
- C PNLJ A: 6 I/Os
- B BNLJ C: 5 I/Os
- C INLJ B: 6 I/Os

Query Optimization - Selinger

- **Pass i** (Repeat until all relations are joined)**:**
  take in list of optimal plans for (i - 1 relations, interesting order) from Pass i-1, and compute minimum cost plan for (i relations, interesting orders) (every size i subset of the n relations)

Pass 3:

- A BNLJ B: 5 I/Os
- C PNLJ A: 6 I/Os
- B BNLJ C: 5 I/Os

- (AB) BNLJ C: 14 I/Os
- (CA) INLJ B: 13 I/Os
- (CA) BNLJ B: 12 I/Os
- (BC) PNLJ A: 13 I/Os

Selectivity Estimation

- To estimate the cost of a query, we add up the estimated costs of each operator in the query
  - Need to know the size of the **intermediate relations** (generated from one operator and passed into another) in order to do this!
    - Need to know the **selectivity** of predicates - what % of tuples are selected by a predicate
- These are all estimates: if we don’t know, we make up a value for it (selectivity = 1/10)

Selectivity Estimation - Equalities

Predicate	Selectivity	Assumption
c = v	1 / (number of distinct values of c in index)	We know |c|.
c = v	1 / 10	We don't know |c|.
c1 = c2	1 / MAX(number of distinct values of c1, number of distinct values of c2)	We know |c1| and |c2|.
c1 = c2	1 / (number of distinct values of ci)	We know |ci| but not |other column|.
c1 = c2	1 / 10	We don't know |c1| or |c2|.

- **|column|** = the number of distinct values for the column
- If you have an index on the column, you can assume you know |column|, max(c), and min(c)
- When applying selectivity to # of records, take the floor of the result. (e.g. 256.3 → 256 records)

Selectivity Estimation - Inequalities on Integers

Predicate	**Selectivity**	**Assumption**
$c < v$ $c > v$	(v - min(c)) / (max(c) - min(c) + 1) (max(c) - v) / (max(c) - min(c) + 1)	We know max(c) and min(c). c is an integer.
$c < v$ $c > v$	1 / 10	We don’t know max(c) and min(c). c is an integer.
$c <= v$ $c >= v$	(v - min(c)) / (max(c) - min(c) + 1) + (1 / |c|) (max(c) - v) / (max(c) - min(c) + 1) + (1 / |c|)	We know max(c) and min(c). c is an integer.
$c <= v$ $c >= v$	1 / 10	We don’t know max(c) and min(c). c is an integer.

* We add 1 to the denominator in order for our [low, high] range to be inclusive.
E.g. range [2, 4] = 2, 3, 4 ⟶ (4 - 2) + 1 = 3

Selectivity Estimation - Inequalities on Floats

Predicate	**Selectivity**	**Assumption**
c >= v	(max(c) - v) / (max(c) - min(c))	We know max(c) and min(c). c is a float.
c >= v	1 / 10	We don't know max(c) and min(c). c is a float.
c <= v	(v - min(c)) / (max(c) - min(c))	We know max(c) and min(c). c is a float.
c <= v	1 / 10	We don't know max(c) and min(c). c is a float.

* We don't add 1 to the denominator. floats are continuous, integers are discrete)
E.g. range [2.0, 4.0] = 2.0, 2.1, …, 3.9, 4.0 ⟶ 4.0 - 2.0 = 2.0

Selectivity Estimation - Connectives

Predicate	**Selectivity**	**Assumption**
p1 **AND** p2	S(p1)*S(p2)	Independent predicates
p1 **OR** p2	S(p1) **+** S(p2) **-** S(p1)*S(p2)	Independent predicates
NOT p	1 **-** S(p)	

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

```
SELECT * FROM R
```

- R(a, b, c) has 1000 tuples
- index on a with 50 unique integer values, uniformly distributed in the range [1, 50]
- index on b with 100 unique float values, uniformly distributed in the range [1, 100]
- no index on c

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

```
SELECT * FROM R
```

1000 tuples
(no predicates, select all)

- R(a, b, c) has 1000 tuples
- index on a with 50 unique integer values, uniformly distributed in the range [1, 50]
- index on b with 100 unique float values, uniformly distributed in the range [1, 100]
- no index on c

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

```
SELECT * FROM R
    WHERE a = 42;
```

- R(a, b, c) has 1000 tuples
- index on a with 50 unique integer values, uniformly distributed in the range [1, 50]
- index on b with 100 unique float values, uniformly distributed in
- no index on c

c = v	1 / (number of distinct values of c in index)	We know |c|.

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

```
SELECT * FROM R
   WHERE a = 42;
```

50 unique values in **a**

1/50 * (1000 tuples) = 20 tuples

- R(a, b, c) has 1000 tuples
- index on a with 50 unique integer values, uniformly distributed in the range [1, 50]
- index on b with 100 unique float values, uniformly distributed in the range [1, 100]
- no index on c

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

```
SELECT * FROM R
   WHERE c = 42;
```

- R(a, b, c) has 1000 tuples
- index on a with 50 unique integer values, uniformly distributed in the range [1, 50]
- index on b with 100 unique float values, uniformly distributed in

c = v	1 / 10	We don’t know |c|.

- no index on c

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

```
SELECT * FROM R
    WHERE c = 42;
```

no information about **c**

1/10 * (1000 tuples) = 100 tuples

- R(a, b, c) has 1000 tuples
- index on a with 50 unique integer values, uniformly distributed in the range [1, 50]
- index on b with 100 unique float values, uniformly distributed in the range [1, 100]
- no index on c

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

```
SELECT * FROM R
   WHERE a <= 25;
```

- R(a, b, c) has 1000 tuples
- index on a with 50 unique integer values, uniformly distributed in the range [1, 50]
- index on b with 100 unique float

c <= v	(v - min(c)) / (max(c) - min(c) + 1) + (1 / |c|)	We know max(c) and min(c). c is an integer.
c >= v	(max(c) - v) / (max(c) - min(c) + 1) + (1 / |c|)	

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

```
SELECT * FROM R
    WHERE a <= 25;
```

Sel(**a <= 25**)

= (25 - 1)/(50 - 1 + 1) + 1/50 = 1/2

1/2 * (1000 tuples) = 500 tuples

- R(a, b, c) has 1000 tuples
- index on a with 50 unique integer values, uniformly distributed in the range [1, 50]
- index on b with 100 unique float values, uniformly distributed in the range [1, 100]
- no index on c

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

```
SELECT * FROM R
    WHERE b <= 25;
```

- R(a, b, c) has 1000 tuples
- index on a with 50 unique integer values, uniformly distributed in the range [1, 50]
- index on b with 100 unique float

c <= v	(v - min(c)) / (max(c) - min(c))	We know max(c) and min(c). c is a float.

- no index on c

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

```
SELECT * FROM R
    WHERE b <= 25;
```

Sel(**b <= 25**)

= (25 - 1)/(100 - 1) = 24/99 = 0.2424...

floor(0.2424... * (1000 tuples)) = 242 tuples

- R(a, b, c) has 1000 tuples
- index on a with 50 unique integer values, uniformly distributed in the range [1, 50]
- index on b with 100 unique float values, uniformly distributed in the range [1, 100]
- no index on c

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

```
SELECT * FROM R
    WHERE a <= 25
      AND b <= 25;
```

- R(a, b, c) has 1000 tuples
- index on a with 50 unique integer values, uniformly distributed in the range [1, 50]
- index on b with 100 unique float values, uniformly distributed in the range [1, 100]

p1 **AND** p2	S(p1)*S(p2)	Independent predicates

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

```
SELECT * FROM R
    WHERE a <= 25
      AND b <= 25;
```

Sel(**a <= 25**) * Sel(**b <= 25**)

= ½ * 24/99 = 0.1212...

floor(0.1212... * (1000 tuples)) = 121 tuples

- R(a, b, c) has 1000 tuples
- index on a with 50 unique integer values, uniformly distributed in the range [1, 50]
- index on b with 100 unique float values, uniformly distributed in the range [1, 100]
- no index on c

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

```
SELECT * FROM R
   WHERE a = c;
```

- R(a, b, c) has 1000 tuples
- index on a with 50 unique integer values, uniformly distributed in the range [1, 50]
- index on b with 100 unique float values, uniformly distributed in

c1 = c2	1 / (number of distinct values of ci)	We know |ci| but not |other column|.

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

```
SELECT * FROM R
    WHERE a = c;
```

no information about **c**

1/50 * (1000 tuples) = 20 tuples

- R(a, b, c) has 1000 tuples
- index on a with 50 unique integer values, uniformly distributed in the range [1, 50]
- index on b with 100 unique float values, uniformly distributed in the range [1, 100]
- no index on c

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

```
SELECT * FROM R
    WHERE R.a = S.a;
```

- R(a, b, c) has 1000 tuples
- index on R.a with 50 unique integer values, uniformly distributed in the range [1, 50]
- S(a) has 500 tuples
- index on S.a with 25 unique

c1 = c2	1 / MAX(number of distinct values of c1, number of distinct values of c2)	We know |c1| and |c2|.

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

```
SELECT * FROM R
    WHERE R.a = S.a;
```

- R(a, b, c) has 1000 tuples
- index on R.a with 50 unique integer values, uniformly distributed in the range [1, 50]
- S(a) has 500 tuples
- index on S.a with 25 unique integer values, uniformly distributed in the range [1, 25]

Sel(**R.a = S.a**)

= 1/MAX(50, 25) = 1/50

1/50 * (1000 tuples * 500 tuples) = 10,000 tuples