Discussion 7

Selectivity and Query Optimization

Announcements

Vitamin 7 (Query Optimization) due **Monday, March 11 at 11:59pm**

Project 3 Part 1 due **Wednesday, March 6 at 11:59PM**

Selectivity Estimation

Selectivity Estimation

- To estimate cost of a query, we add up the estimated costs of each operator (step) in the query
  - Need to know the size of the **intermediate relations** (table generated between operators) in order to do this!
    - Need **selectivity** of predicates - what % of tuples are selected by a predicate - to estimate intermediate relation's size

NOTE: These are all estimates: if we don't know, approximate (we use **selectivity = 1/10** in this class as a default)

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.

NOTICE: We add 1 to the denominator in order for our [low, high] range to be inclusive.
E.g. range [2, 4] = 2, 3, 4 $\rightarrow$ (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.

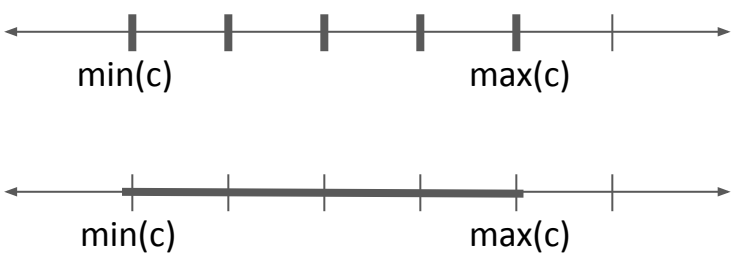
NOTICE: We don't add 1 to the denominator. floats are continuous, integers are discrete)

E.g. range [2.0, 4.0] = 2.0, 2.1, …, 3.9, 4.0 ⟶ 4.0 - 2.0 = 2.0

Selectivity Estimation - Inequalities Ints vs Floats

Predicate	Selectivity	Assumption
c >= v	(max(c) - v) / (max(c) - min(c) + 1)	We know max(c) and min(c). c is an integer.
c >= v	(max(c) - v) / (max(c) - min(c))	We know max(c) and min(c). c is a float.

With floats, need to account for continuous area between min(c) and max(c). Assume min is 2 and max is 4, then length of range is 4 - 2 = 2.



With ints, we need to account for min(c) and max(c) inclusive. Assume min is 2 and max is 4, then have a total of 4 - 2 + 1 = 3 possible values.

Selectivity Estimation - Connectives

Predicate	Selectivity	Assumption
p1 **AND** p2	S(p1)*S(p2)	Independent predicates
p1 **OR** p2	S(p1) **+** S(p2) **-** S(p1 AND p2)	
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
- columns are independent

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
- columns are independent

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 the range [1, 100]
- no index on c
- columns are independent

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
- columns are independent

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 the range [1, 100]
- no index on c
- columns are independent

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
- columns are independent

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 values, uniformly distributed in the range [1, 100]
- no index on c
- columns are independent

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
- columns are independent

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 values, uniformly distributed in the range [1, 100]
- no index on c
- columns are independent

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
- columns are independent

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

```
SELECT * FROM R
    WHERE c <= 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]
- no index on c
- columns are independent

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

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

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
- columns are independent

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]
- no index on c
- columns are independent

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
- columns are independent

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

```
SELECT * FROM R
   WHERE a <= 25
     AND c <= 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]
- no index on c
- columns are independent

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

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

Sel(**a <= 25**) * Sel(**c <= 25**)
= ½ * 1/10 = 1/20
1/20 * (1000 tuples) = 50 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
- columns are independent

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

```
SELECT * FROM R
   WHERE a <= 25
     AND a > 10;
```

- 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
- columns are independent

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

```
SELECT * FROM R
    WHERE a <= 25
      AND a > 10;
```

sel(10 < a <= 25)
= (25 - 10) / 50 = 0.3
0.3 * (1000 tuples) = 300 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
- columns are independent

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

```
SELECT * FROM R
   WHERE a <= 25
      OR 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]
- no index on c
- columns are independent

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

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

Sel(**a <= 25**) + Sel(**b <= 25**)
- Sel(**a <= 25**) * Sel(**b <= 25**)
  = ½ + 24/99 - ½ * 24/99
  = 0.62121...

0.62121... * (1000 tuples) = 621 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
- columns are independent

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 the range [1, 100]
- no index on c
- columns are independent

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
- columns are independent

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

```
SELECT * FROM R,S
   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]

Selectivity Estimation - Worksheet

How many tuples are selected by the following query?

```
SELECT * FROM R,S
    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

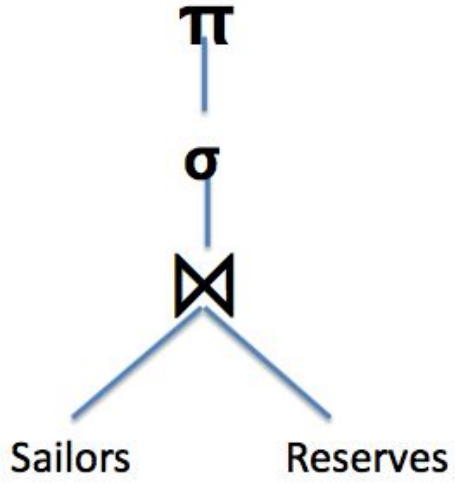
Query Optimization

Iterators

- Recall: relational operators operate on relations and return relations
- We can implement this as: operate on an iterator (of the input relation) and return an iterator (of the output)
  - Optionally choose if we wish to **materialize** the output relation (write it to disk) or **stream** it to the next operator

Query Optimization - Background

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



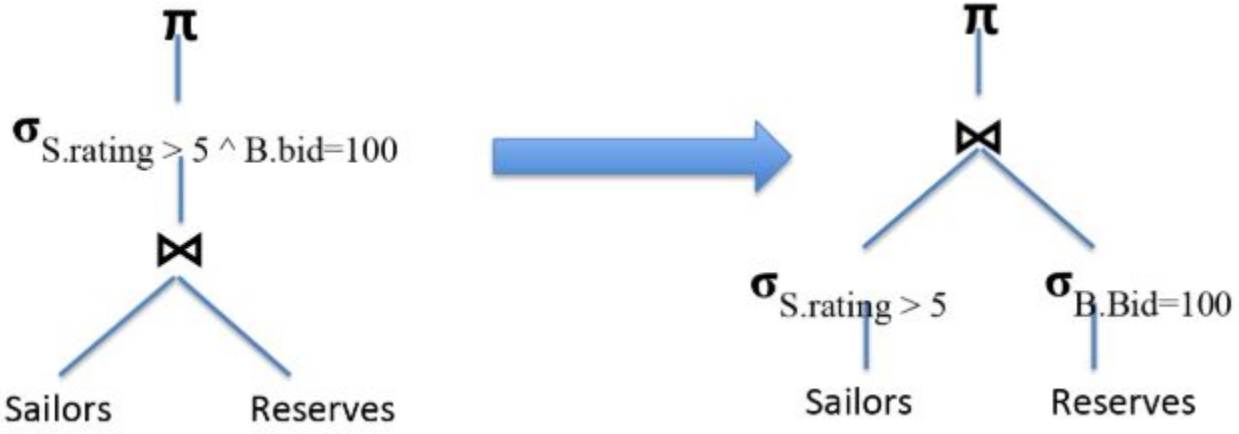
π…(σ…(Sailors ⋈… Reserves))

σ…(π…(Sailors ⋈… Reserves))

s

Query Optimization - Alternate Plans

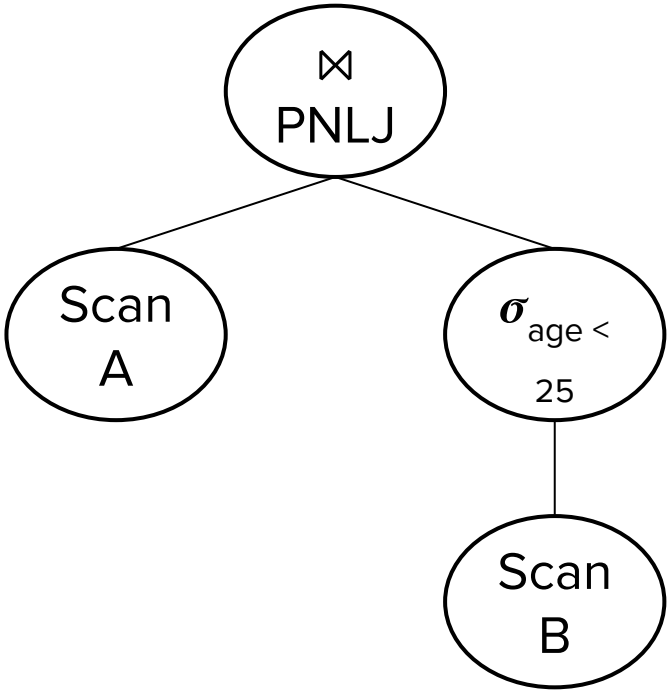
- Given a plan, some things we can do are:
  - **Push selections/projections down the tree**
    - The earlier we reduce the size of our input data, the fewer I/Os are incurred as we traverse up the tree
    - Only affects I/O cost if materialized, or if operator only makes one pass (so not right relation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



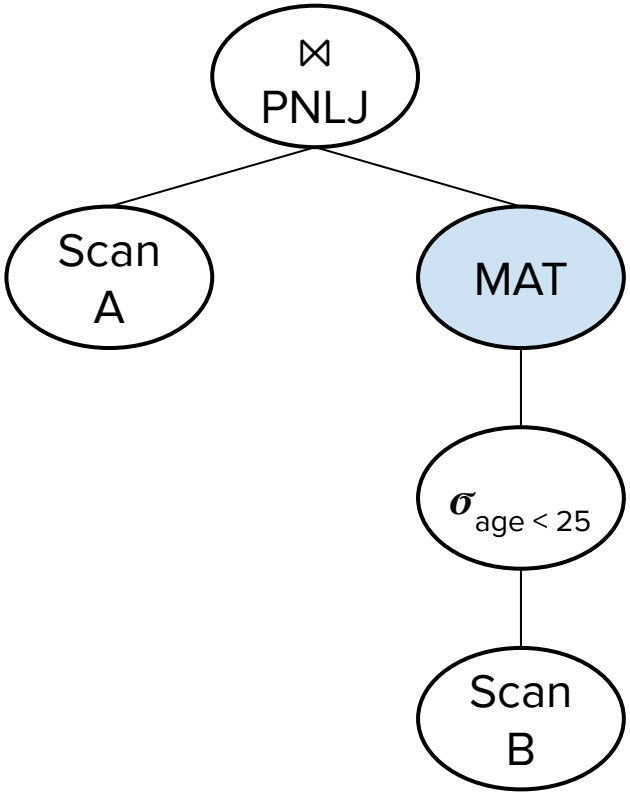
- Table A: takes 50 I/Os to perform a scan
- Table B: takes 100 I/Os to perform a scan
- Sel(B.age < 25) = 0.5, [B] = 100

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

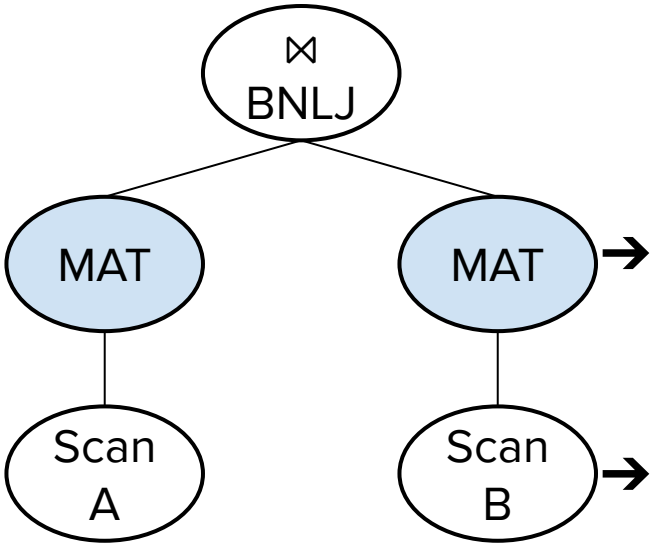
By materializing the intermediate relation, we're applying $\sigma_{age < 25}$ before PNLJ, and performing the join on the *result* of the selection.

Cost = *Scan A* (50) + *Scan B* (100)
+ *Materialize* (100 * 0.5) + *PNLJ* (50 * 50)
➔ 2,700 I/Os in total

Query Optimization - Caution When Calculating Join Costs

- We **cannot blindly apply** the join cost formulas
- Join cost depends on whether previous operators materialize intermediate relations OR stream them in as input
- Remember query optimization involves optimizing for the cheapest *estimated* query plan across ALL passes; therefore, previous passes must be taken into account

Query Optimization - Join Considerations



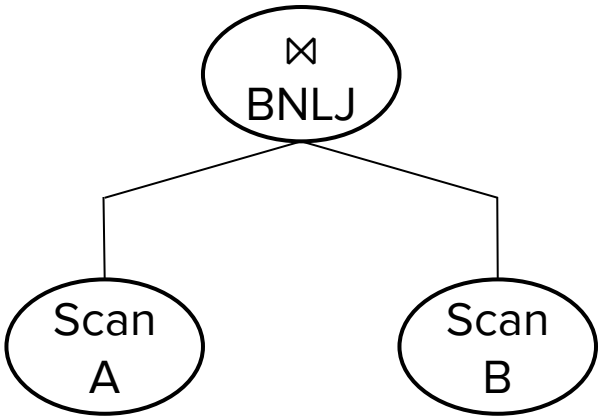
- Table A: takes 50 I/Os to perform a scan
- Table B: takes 100 I/Os to perform a scan
- B = 5 buffer pages

➔ Cost of performing this query plan involves cost of scanning A & B + cost of materializing intermediate relations + cost of joining A & B

➔ Cost = (50 + 100) + (50 + 100) + 50 + ⌈50/3⌉*100 = **2050 I/Os**

➔ The last 2 terms come from the BNLJ formula, and previous terms come from previous operators

Query Optimization - Join Considerations



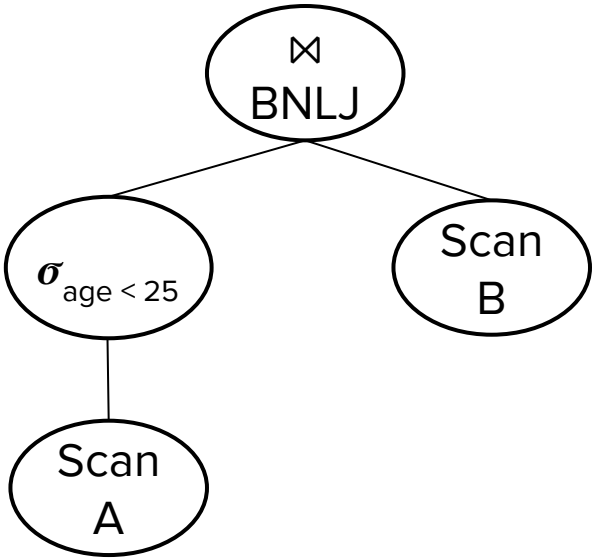
- Table A: takes 50 I/Os to perform a scan
- Table B: takes 100 I/Os to perform a scan
- B = 5 buffer pages

➔ Cost of performing this query plan involves cost of joining A & B, which includes scan cost

➔ Cost = 50 + ⌈50/3⌉*100 = **1750 I/Os**

➔ Here we apply the BNLJ formula directly since there is no materialization and no other operators that might reduce the number of pages provided as input to the BNLJ operator

Query Optimization - Join Considerations



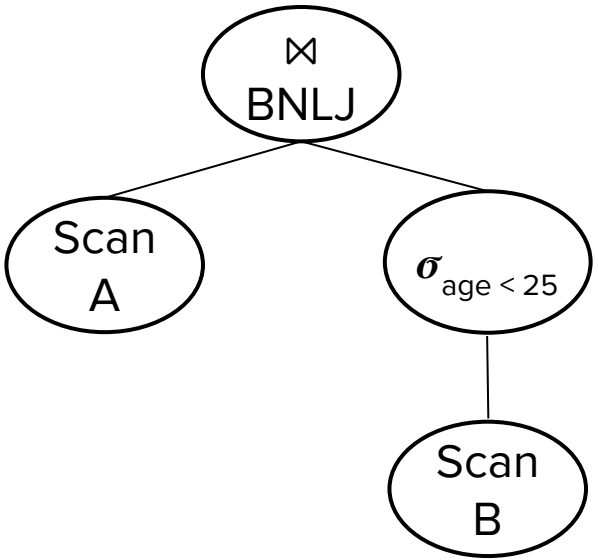
- Table A: takes 50 I/Os to perform a scan
- Table B: takes 100 I/Os to perform a scan
- B = 5 buffer pages
- Sel(A.age < 25) = 0.5

➔ Cost of performing this query plan involves cost of joining A & B, which includes scan cost, and also considering how many pages of A are provided as input to the BNLJ operator

➔ Cost = 50 + ⌈25/3⌉*100 = **950 I/Os**

➔ Only 25 pages from A move on to the join, so the number of times B must be scanned decreases (the last term)

Query Optimization - Join Considerations



- Table A: takes 50 I/Os to perform a scan
- Table B: takes 100 I/Os to perform a scan
- B = 5 buffer pages
- Sel(B.age < 25) = 0.5

➔ Cost of performing this query plan involves cost of joining A & B, which includes scan cost

➔ Cost = 50 + ⌈50/3⌉*100 = **1750 I/Os**

➔ Since B is not materialized, each time B is scanned (which happens ⌈50/3⌉ times) selection occurs on-the-fly

➔ ***Takeaway***: Pushing down selections has different impacts on outer vs inner relation in BNLJ

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)

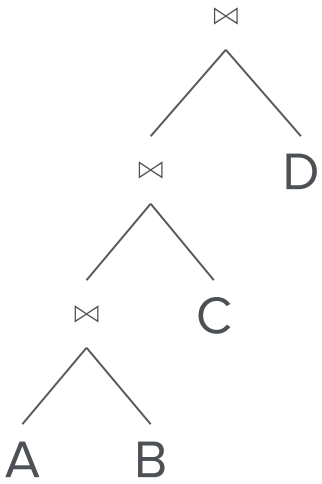
Query Optimization - Selinger

- 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
- System R does not perform any materialization by default

Query Optimization - Selinger

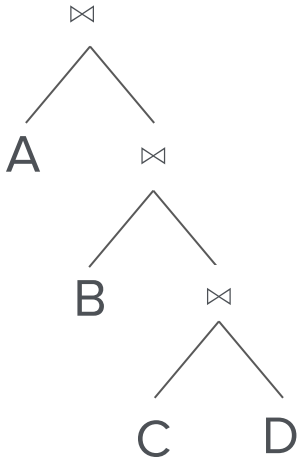
- Why only left-deep trees?
  - Join new tables one at a time from the right
  - Create an ordering in which to add tables to the query being executed
  - Too many possible trees for joins
    - Using only left-deep trees: N! different ways to order relations
    - Including all permutations tree layouts: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

Query Optimization - Selinger



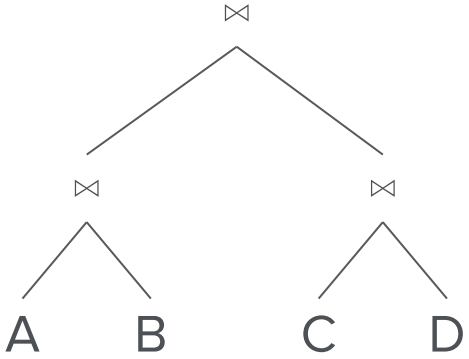
Left-deep

((A ⋈ B) ⋈ C) ⋈ D



Right-deep

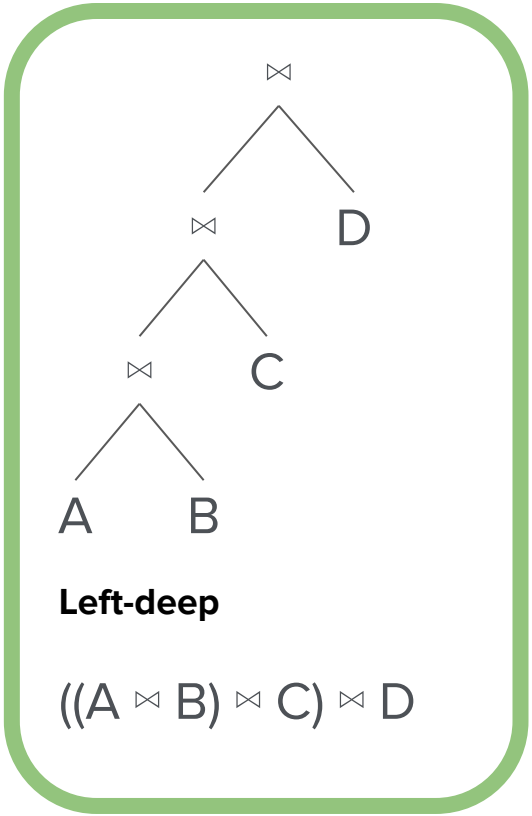
A ⋈ (B ⋈ (C ⋈ D))



Bushy

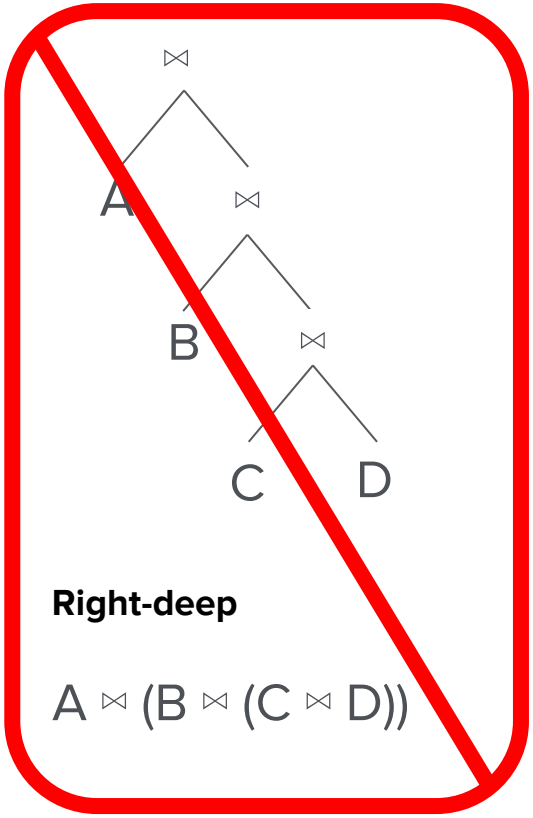
(A ⋈ B) ⋈ (C ⋈ D)

Only consider left-deep plans!



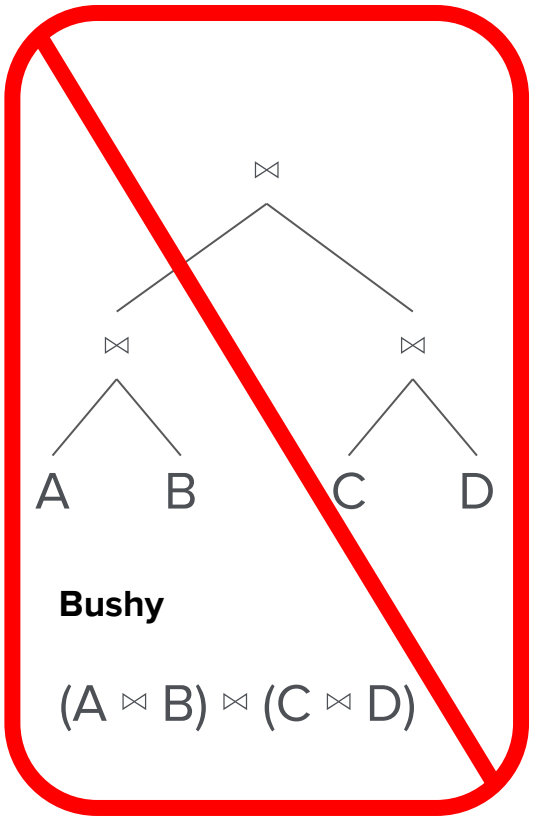
Left-deep

((A ⋈ B) ⋈ C) ⋈ D



Right-deep

A ⋈ (B ⋈ (C ⋈ D))



Bushy

(A ⋈ B) ⋈ (C ⋈ 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 pass, 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 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 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

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

Query Optimization - Worksheet

Consider the relations R(a, b), S(b, c), and T(c, d) with:

- Alt 2 clustered indexes on R.b, S.b, T.c, and T.d
- Alt 2 unclustered index on R.a

Assume it takes 2 I/Os to reach the level above a leaf node and that no index or data pages are ever cached. All indexes have 100 unique integer index keys in the range [1, 100].

```
SELECT *

FROM R, S, T

WHERE R.b = S.b and S.c = T.c

AND R.a <= 50;
```

Query Optimization - Worksheet

Assume:

- R has 1000 data pages, 10000 records
- The index on R.a has 50 leaf pages
- The index on R.b has 100 leaf pages

1. How many IOs does a full scan on R take?
2. How many IOs does an index scan on R.a take?
3. How many IOs does an index scan on R.b take?
4. How many pages from R will advance to the next stage for all of these access plans?

Query Optimization - Worksheet

Assume:

- R has 1000 data pages, 10000 records
- The index on R.a has 50 leaf pages
- The index on R.b has 100 leaf pages

1. How many IOs does a full scan on R take?

   1000 IOs, need to read every data page

2. How many IOs does an index scan on R.a take?

   Sel(R.a <= 50) = (50 - 1) / (100- 1 + 1) + (1/100) = ½

   2 + ½*(50 leaf pages) + ½*(10000 records) = **5,027 IOs**

Query Optimization - Worksheet

Assume:

- R has 1000 data pages, 10000 records
- The index on R.a has 50 leaf pages
- The index on R.b has 100 leaf pages

3. How many IOs does an index scan on R.b take?

   No single table predicates on R.b, so we have to read in everything

   2 + (100 leaf pages) + (1000 data pages) = **1102 IOs**

4. How many pages from R will advance to the next stage for all of these access plans?

   Sel(R.a <= 50) = ½ and 10000 records / 1000 data pages = 10 records / page in table R

   ½*(10000 records) = 5000 records

   5000 records / 10 records per page = **500 pages**

Query Optimization - Worksheet

Assume the *other* single table access plans have the following IO costs:

- Full scan on S: 2000 I/Os
- Index scan on S.b: 2500 I/Os
- Full scan on T: 3000 I/Os
- Index scan on T.c 3500 I/Os
- Index scan on T.d: 3500 I/Os

I/O costs from #1-4:

- Full scan on R: 1000 I/Os
- Index scan on R.a: 5207 I/Os
- Index scan on R.b: 1102 I/Os

Which single table access plans advance to the next stage?

Query Optimization - Worksheet

- **Full scan on S: 2000 I/Os**
- **Index scan on S.b: 2500 I/Os**
- **Full scan on T: 3000 I/Os**
- **Index scan on T.c 3500 I/Os**
- Index scan on T.d: 3500 I/Os

I/O costs from #1-4:

- **Full scan on R: 1000 I/Os**
- Index scan on R.a: 5207 I/Os
- **Index scan on R.b: 1102 I/Os**

Full scan on R (best overall plan for R)

Index scan on R.b (interesting order because R.b is used in a join)

Full scan on S (best overall plan for S)

Index scan on S.b (interesting order because S.b is used in a join)

Full scan on T (best overall plan for T)

Index scan on T.c (produces interesting order on T.c)

Query Optimization - Worksheet

With the tables:

R: 1000 data pages, 10000 records

S: 2000 data pages, 40000 records

T: 3000 data pages, 30000 records

a. R BNLJ S

   i. Which single table access plans for R and S will minimize the cost of this join?

Query Optimization - Worksheet

With the tables:

R: 1000 data pages, 10000 records

S: 2000 data pages, 40000 records

T: 3000 data pages, 30000 records

a. R BNLJ S

   i. Which single table access plans for R and S will minimize the cost of this join?

Since this is a (block) nested loop join, the order of the tuples does not reduce the I/O cost. Therefore, the cheapest single table access plans will be chosen. This means a full scan will be used for both R and S, since those are the best overall plans that advanced from the last stage.

Query Optimization - Worksheet

With the tables:

R: 1000 data pages, 10000 records

S: 2000 data pages, 40000 records

T: 3000 data pages, 30000 records

a. R BNLJ S

ii. What is the I/O cost for R join S? Assume we have 52 buffer pages.

Query Optimization - Worksheet

With the tables:

R: 1000 data pages, 10000 records

S: 2000 data pages, 40000 records

T: 3000 data pages, 30000 records

a. R BNLJ S

ii. What is the I/O cost for R join S? Assume we have 52 buffer pages.

[R] + ceil([R (after selection)] / B-2) * [S] = 1000 + ceil(500 / 50) * 2000 = **21,000**

We have an initial cost of 1000 I/Os to do a full scan on R, but since we push down the selection on R.a, only 500 pages of R will be passed along to the BNLJ operator. This means that the rest of the join will cost (500/50)(2000), since we scan all of S for each chunk from R of size 500/50 = 10.

Query Optimization - Worksheet

With the tables:

R: 1000 data pages, 10000 records

S: 2000 data pages, 40000 records

T: 3000 data pages, 30000 records

b. R SMJ S

   i. Which single table access plans for R and S will minimize the cost of this join?

Query Optimization - Worksheet

With the tables:

R: 1000 data pages, 10000 records

S: 2000 data pages, 40000 records

T: 3000 data pages, 30000 records

b. R SMJ S

   i. Which single table access plans for R and S will minimize the cost of this join?

If the Sort Merge Join operator is passed the pages of R and S already sorted on the columns R.b and S.b respectively, then the sort part of SMJ can be avoided. Therefore, the query optimizer will perform index scans on R.b and S.b before performing the SMJ.

Query Optimization - Worksheet

With the tables:

R: 1000 data pages, 10000 records

S: 2000 data pages, 40000 records

T: 3000 data pages, 30000 records

b. R SMJ S

ii. What is the I/O cost for R join S? Assume we have 52 buffer pages.

Query Optimization - Worksheet

With the tables:

R: 1000 data pages, 10000 records

S: 2000 data pages, 40000 records

T: 3000 data pages, 30000 records

b. R SMJ S

ii. What is the I/O cost for R join S? Assume we have 52 buffer pages.

sort(R) + sort(S) + ([R] + [S]) = 0 + 0 + (1102 + 2500) = **3602 I/Os**

No need to sort either table because index scans on R.b and S.b would result in sorted order already. Thus, the cost is just to merge, which involves simply scanning the tuples.

Query Optimization - Worksheet

Assume the rest of the join IO costs are as follows

1. R BNLJ S: 1.a IOs
2. R SMJ S: 1.b IOs
3. S BNLJ R: 18,000 IOs
4. S SMJ R: 3,000 IOs
5. R BNLJ T: 30,000 IOs
6. R SMJ T: 40,000 IOs
7. T BNLJ R: 35,000 IOs
8. T SMJ R: 20,000 IOs
9. S BNLJ T: 15,000 IOs
10. S SMJ T: 10,000 IOs
11. T BNLJ S: 25,000 IOs
12. T SMJ S: 30,000 IOs

Which of these joins will actually be considered by the query optimizer on pass 2?

Query Optimization - Worksheet

Assume the rest of the join IO costs are as follows

1. R BNLJ S: 1.a IOs
2. R SMJ S: 1.b IOs
3. S BNLJ R: 18,000 IOs
4. S SMJ R: 3,000 IOs
5. R BNLJ T: 30,000 IOs
6. R SMJ T: 40,000 IOs
7. T BNLJ R: 35,000 IOs
8. T SMJ R: 20,000 IOs
9. S BNLJ T: 15,000 IOs
10. S SMJ T: 10,000 IOs
11. T BNLJ S: 25,000 IOs
12. T SMJ S: 30,000 IOs

Which of these joins will actually be considered by the query optimizer on pass 2?

Don’t consider cross joins (any joins involving R and T, for this query)

Query Optimization - Worksheet

Assume the rest of the join IO costs are as follows

1. R BNLJ S: 1.a IOs
2. R SMJ S: 1.b IOs
3. S BNLJ R: 18,000 IOs
4. S SMJ R: 3,000 IOs
5. ~~R BNLJ T: 30,000 IOs~~
6. ~~R SMJ T: 40,000 IOs~~
7. ~~T BNLJ R: 35,000 IOs~~
8. ~~T SMJ R: 20,000 IOs~~
9. S BNLJ T: 15,000 IOs
10. S SMJ T: 10,000 IOs
11. T BNLJ S: 25,000 IOs
12. T SMJ S: 30,000 IOs

Which of these joins will advance to the next pass of the query optimizer?

Query Optimization - Worksheet

Assume the rest of the join IO costs are as follows

1. R BNLJ S: 1.a IOs
2. R SMJ S: 1.b IOs
3. S BNLJ R: 18,000 IOs
4. S SMJ R: 3,000 IOs
5. ~~R BNLJ T: 30,000 IOs~~
6. ~~R SMJ T: 40,000 IOs~~
7. ~~T BNLJ R: 35,000 IOs~~
8. ~~T SMJ R: 20,000 IOs~~
9. S BNLJ T: 15,000 IOs
10. S SMJ T: 10,000 IOs
11. T BNLJ S: 25,000 IOs
12. T SMJ S: 30,000 IOs

Which of these joins will advance to the next pass of the query optimizer?

None of the joins produce an interesting order (no downstream joins, ORDER BY, GROUP BY). Only consider best join for each considered set of tables

Query Optimization - Worksheet

Assume the rest of the join IO costs are as follows

1. R BNLJ S: 1.a IOs
2. R SMJ S: 1.b IOs
3. S BNLJ R: 18,000 IOs
4. **S SMJ R: 3,000 IOs**
5. R BNLJ T: 30,000 IOs
6. R SMJ T: 40,000 IOs
7. T BNLJ R: 35,000 IOs
8. T SMJ R: 20,000 IOs
9. S BNLJ T: 15,000 IOs
10. **S SMJ T: 10,000 IOs**
11. T BNLJ S: 25,000 IOs
12. T SMJ S: 30,000 IOs

Will any of these remaining joins produce an interesting order?

Query Optimization - Worksheet

Assume the rest of the join IO costs are as follows

1. R BNLJ S: 1.a IOs
2. R SMJ S: 1.b IOs
3. S BNLJ R: 18,000 IOs
4. **S SMJ R: 3,000 IOs**
5. R BNLJ T: 30,000 IOs
6. R SMJ T: 40,000 IOs
7. T BNLJ R: 35,000 IOs
8. T SMJ R: 20,000 IOs
9. S BNLJ T: 15,000 IOs
10. **S SMJ T: 10,000 IOs**
11. T BNLJ S: 25,000 IOs
12. T SMJ S: 30,000 IOs

Will any of these remaining joins produce an interesting order?

No, S SMJ R uses column b (not interesting), S SMJ T uses column c (not interesting)

Query Optimization - Worksheet

How could we modify the query so that the R SMJ S produces an interesting order?

Query Optimization - Worksheet

How could we modify the query so that the R SMJ S produces an interesting order?

R SMJ S will be sorted on column b so we need b to be interesting. We could add ORDER BY b, GROUP BY b, or another join condition involving R.b or S.b to the query to make it interesting.

Query Optimization - Worksheet

Will the query plan: T BNLJ (R SMJ S) be considered by the final pass of the query optimizer?

Query Optimization - Worksheet

Will the query plan: T BNLJ (R SMJ S) be considered by the final pass of the query optimizer?

No, this query plan is **not** left-deep (all join results must be on the left side of their parent join), so it is not considered in the final pass.

Attendance Link

https://cs186berkeley.net/attendance