Relational Query Optimization I: The Plan Space

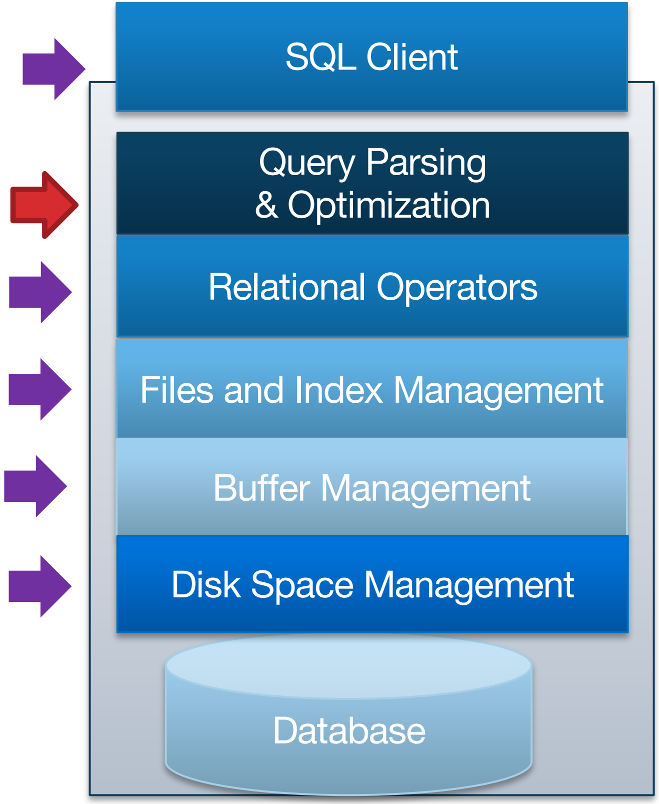
Alvin Cheung

Aditya Parameswaran

R&G 15

Architecture of a DBMS

- Completed
- You are here
- Completed
- Completed
- Completed
- Completed



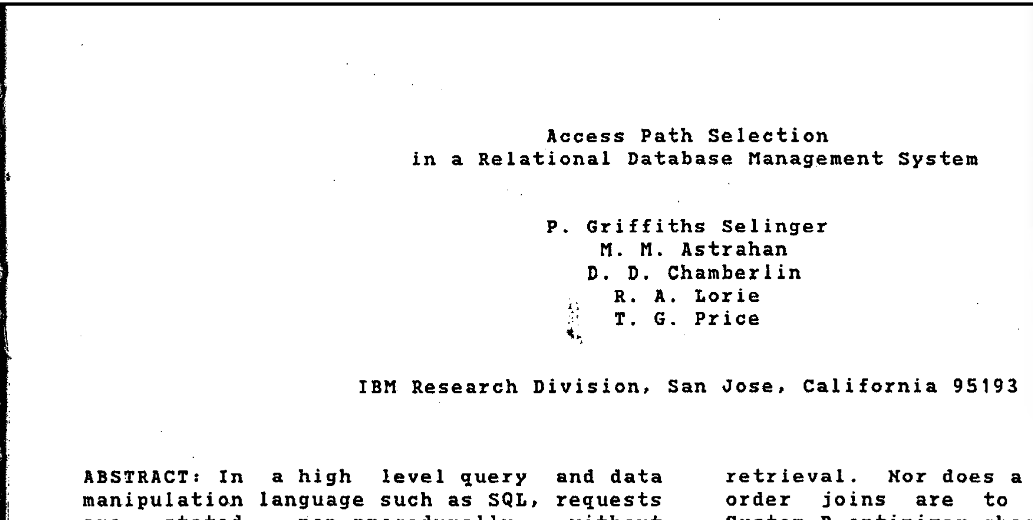
Query Optimization is Magic

- The bridge between a *declarative* domain-specific language…
  - “What” you want as an answer
- … and custom *imperative* computer programs
  - “How” to compute the answer

- A lot of smart people and a lot of time has been spent on this problem!
- Reminiscent of many cutting-edge “AI” problems
  - Similar tricks: optimization + heuristic pruning
  - Analogous to *AI-based Software Synthesis*

Invented in 1979 by Pat Selinger et al.

- We'll focus on "System R" ("Selinger") optimizers
  - From IBM Research in Almaden
- "Cascades" optimizer is the other common one
  - Later, with notable differences, but similar big picture



Access Path Selection
in a Relational Database Management System

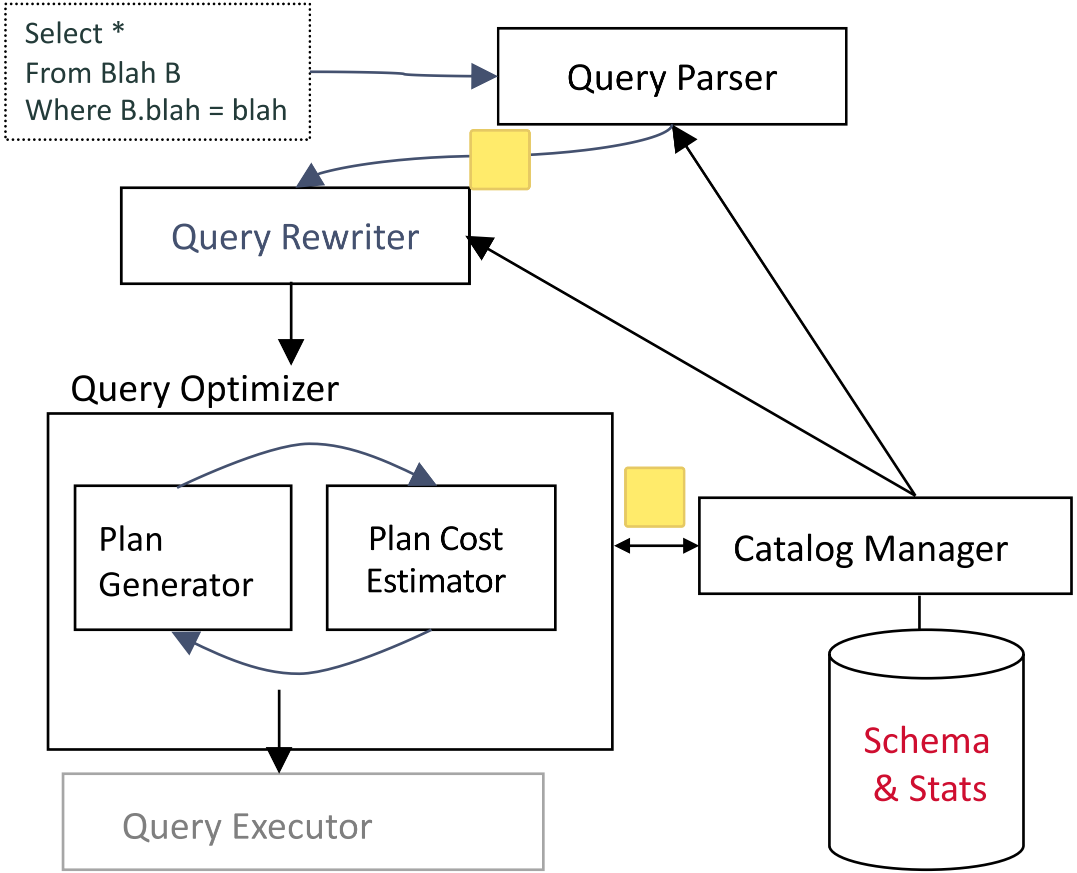
P. Griffiths Selinger
M. M. Astrahan
D. D. Chamberlin
R. A. Lorie
T. G. Price

IBM Research Division, San Jose, California 95193

ABSTRACT: In a high level query and data manipulation language such as SQL, requests

retrieval. Nor does a user specify in what order joins are to be performed. The

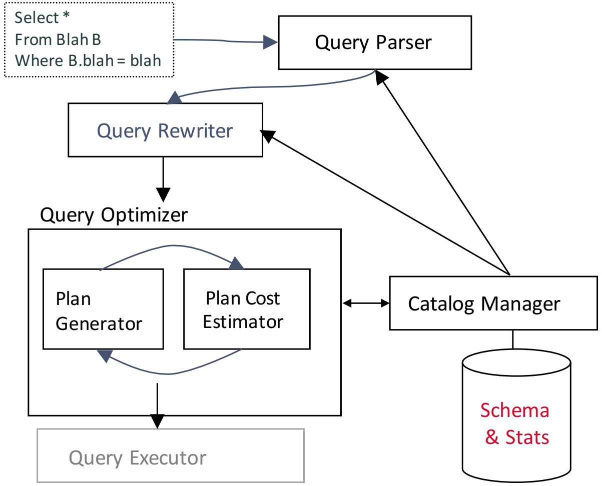
Query Parsing & Optimization: Query Lifecycle



Select *
From Blah B
Where B.blah = blah

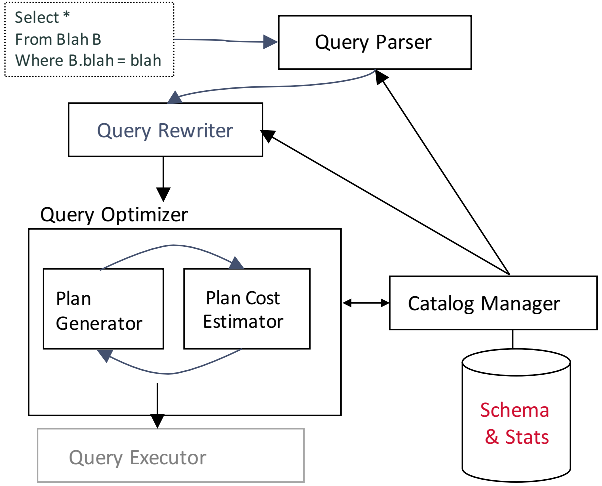
Query Parsing & Optimization Part 2

- Query parser
  - Checks correctness, authorization
  - Generates a parse tree
  - Straightfoward
  - Not our focus



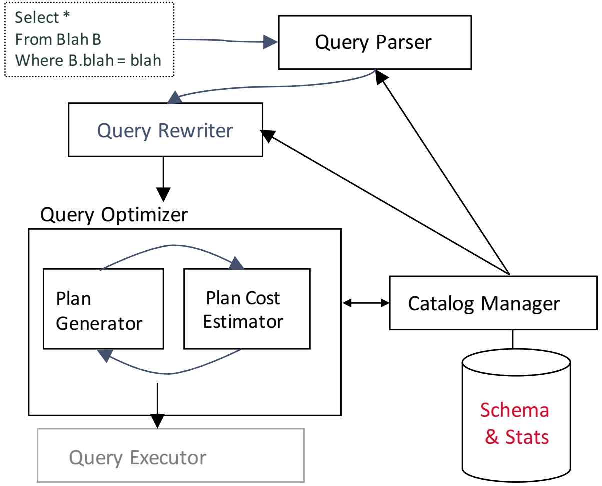
Query Parsing & Optimization Part 3

- Query rewriter
  - Converts queries to canonical form
    - flatten views
    - subqueries into fewer query blocks
      - e.g., by replacing w/ joins
  - Not our focus



Query Parsing & Optimization Part 4

- “Cost-based” Query Optimizer
    - Our focus!
    - Optimizes 1 query block at a time
        - Select, Project, Join
        - GroupBy/Agg
        - Order By (if top-most block)
    - Uses catalog stats to find least-“cost” plan per query block
    - Often not truly “optimal”, lots of heuristic rules and magic



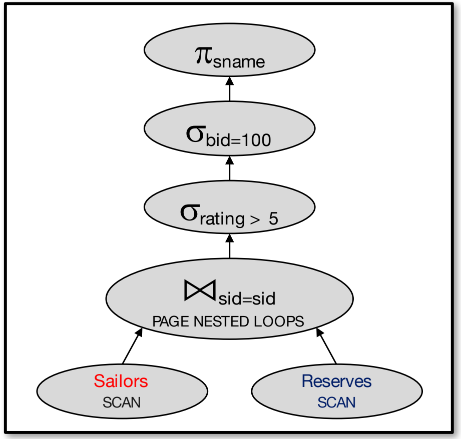
Query Optimization Overview

- Query block can be converted to relational algebra
- Relational algebra can be represented as exp. tree
- Each operator has implementation choices
- Operators can also be applied in different orders!

```
SELECT S.sname
  FROM Reserves R, Sailors S
 WHERE R.sid=S.sid
   AND R.bid=100
   AND S.rating>5
```

$$\pi_{(sname)}\sigma_{(bid=100 \wedge rating > 5)}$$

$$(Reserves \bowtie Sailors)$$



Query Optimization: The Components

- Three (mostly) orthogonal concerns:
  - Plan space:
    - for a given query, what plans are considered?
    - larger the plan space, more likely to find a cheaper plan, but harder to search
  - Cost estimation:
    - how is the cost of a plan estimated?
    - want to find the cheapest plan
  - Search strategy:
    - how do we “search” in the “plan space”?

Query Optimization: The Goal

- Optimization goal:
    - Ideally: Find the plan with least actual cost = one that runs fastest
    - Reality: Find the plan with least estimated cost.
        - At the very least, try to avoid really bad actual plans!

Today

- We will get a feel for the plan space
- Explore one simple example query

Plan Space

- To generate a space of candidate plans, we need to think about how to rewrite relational algebra expressions into other ones
- Therefore, need a set of equivalence rules

Relational Algebra Equivalences: Selections

- Selections:
  - $\sigma_{c1 \wedge \ldots \wedge cn}(R) \equiv \sigma_{c1}(\ldots(\sigma_{cn}(R))\ldots)$ (cascade)
    - Intuitively, RHS says check c_n first on all tuples, then c_{n-1} etc.
  - $\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$ (commute)

Relational Algebra Equivalences: Projections

- Selections:
  - $\sigma_{c1\wedge\ldots\wedge cn}(R) \equiv \sigma_{c1}(\ldots(\sigma_{cn}(R))\ldots)$ (cascade)
  - $\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$ (commute)
- Projections:
  - $\pi_{a1}(\ldots(R)\ldots) \equiv \pi_{a1}(\ldots(\pi_{a1,\ \ldots,\ an-1}(R))\ldots)$ (cascade)
    - Essentially, allows partial projection earlier in the expression
    - As long as we're keeping a_1 (and everything else we need outside) we're OK
  - Q: Are there any commute rules for projections?

Relational Algebra Equivalences: Cartesian Product

- Selections:
  - $\sigma_{c1 \wedge \ldots \wedge cn}(R) \equiv \sigma_{c1}(\ldots(\sigma_{cn}(R))\ldots)$ (cascade)
  - $\sigma_{c1}(\sigma_{c2}(R)) \equiv \sigma_{c2}(\sigma_{c1}(R))$ (commute)
- Projections:
  - $\pi_{a1}(\ldots(R)\ldots) \equiv \pi_{a1}(\ldots(\pi_{a1, \ldots, an-1}(R))\ldots)$ (cascade)
- Cartesian Product
  - $R \times (S \times T) \equiv (R \times S) \times T$ (associative)
  - $R \times S \equiv S \times R$ (commutative)
    - Recall that the ordering of attributes doesn’t matter

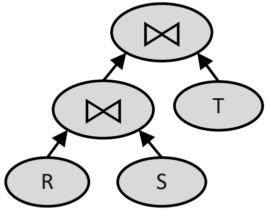
Are Joins Associative and Commutative?

- After all, just Cartesian Products with Selections
- You can think of them as associative and commutative…
- …But beware of join turning into cross-product!
  - Consider R(a,z), S(a,b), T(b,y)
  - Attempt 1: Does this work? Why?
    - $(S \bowtie_{S.b=T.b} T) \bowtie_{S.a=R.a} R \not\equiv S \bowtie_{S.b=T.b} (T \bowtie_{S.a=R.a} R)$
      - *not* legal!! Join on a not permissible
  - Attempt 2: Does this work? Why?
    - $(S \bowtie_{S.b=T.b} T) \bowtie_{S.a=R.a} R \not\equiv S \bowtie_{S.b=T.b} (T \times R)$
      - *not* the same!! No condition for a being same
  - Attempt 3: Does this work?
    - $(S \bowtie_{S.b=T.b} T) \bowtie_{S.a=R.a} R \equiv S \bowtie_{S.b=T.b \wedge S.a=R.a} (T \times R)$

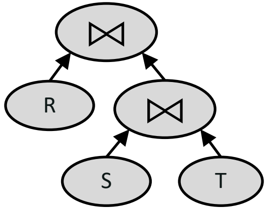
Join ordering, again

- Similarly, note that some join orders have cross products, some don't
- Equivalent for the query above:

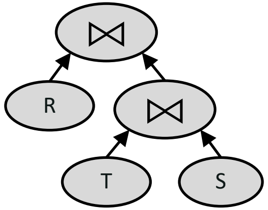
```
SELECT *
  FROM R, S, T
 WHERE R.a = S.a
   AND S.b = T.b;
```



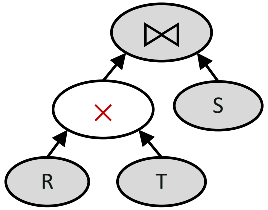
$(R \bowtie_{S.a=R.a} S) \bowtie_{S.b=T.b} T$



$R \bowtie_{S.a=R.a} (S \bowtie_{S.b=T.b} T)$



$R \bowtie_{S.a=R.a} (T \bowtie_{S.b=T.b} S)$



$(R \times T) \bowtie_{S.a=R.a \wedge S.b=T.b} S$

Plan Space

- To generate a space of candidate plans, we need to think about how to rewrite relational algebra expressions into other ones
- Therefore, need a set of equivalence rules – **done**

- Next, will discuss a set of heuristics that are used to restrict attention to plans that are mostly better:
    - we’ve already seen one of these in the relational alg lectures.

Some Common Heuristics: Selections

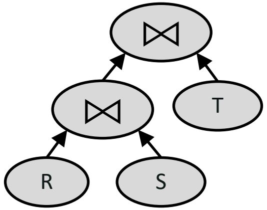
- Selection cascade and pushdown
  - Apply selections as soon as you have the relevant columns
  - Ex:
    - $\pi_{\text{sname}} (\sigma_{(\text{bid}=100 \wedge \text{rating} > 5)} (\text{Reserves} \bowtie_{\text{Reserves.sid}=\text{Sailors.sid}} \text{Sailors}))$
    - $\pi_{\text{sname}} (\sigma_{\text{bid}=100} (\text{Reserves}) \bowtie_{\text{Reserves.sid}=\text{Sailors.sid}} \sigma_{\text{rating} > 5} (\text{Sailors}))$
  - Why is this an improvement?
    - Selection is essentially free, joins are expensive
    - Take care of selections early -- side effect is that the intermediate inputs to joins are smaller

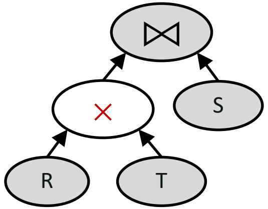
Some Common Heuristics: Projections

- Projection cascade and pushdown
  - Keep only the columns you need to evaluate downstream operators
  - Reserves(sid, bid, day), Sailors (sid, rating, sname)
  - Ex:
    - $\pi_{sname}\sigma_{(bid=100 \wedge rating > 5)}$ (Reserves $\bowtie_{Reserves.sid=Sailors.sid}$ Sailors)
    - Q: How might we cascade and push projections and selections down?
    - π_{sname} (π_{sid}($\sigma_{bid=100}$ (Reserves)) $\bowtie_{Reserves.sid=Sailors.sid}$ $\pi_{sname,sid}$ ($\sigma_{rating > 5}$ (Sailors)))
    - Other rewritings exist! (reorder selection and projection)

Some Common Heuristics

- Avoid Cartesian products
    - Given a choice, do theta-joins rather than cross-products
    - Consider R(a,b), S(b,c), T(c,d)
    - Favor (R ⋈ S) ⋈ T over (R × T) ⋈ S
    - Case where this doesn't quite improve things:
        - if R x T is small (e.g., R & T are very small and S is relatively large)
        - Still, it's a good enough heuristic that we will use it





Plan Space

- To generate a space of candidate plans, we need to think about how to rewrite relational algebra expressions into other ones
- Therefore, need a set of equivalence rules – **done**
- Next, will discuss a set of heuristics that are used to restrict attention to plans that are mostly better – **done**
- Both of these were logical equivalences, will also quickly discuss physical equivalences, next.

Physical Equivalences

- Base table access
  - Heap scan
  - Index scan (if available on referenced columns)
- Equijoins
  - Block (Chunk) Nested Loop: simple, exploits extra memory
  - Index Nested Loop: often good if 1 rel small and the other indexed properly
  - Sort-Merge Join: good with small memory, equal-size tables
  - Grace/Hybrid Hash Join: even better than sort with 1 small table
- Non-Equijoins
  - Block (Chunk) Nested Loop

Schema for Examples

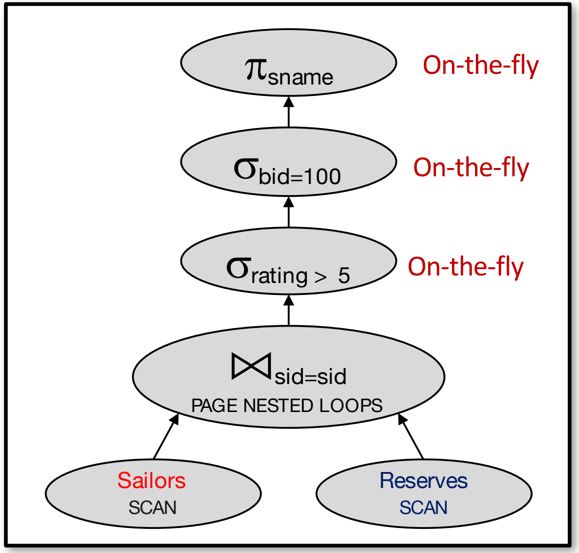
Sailors (*sid*: integer, *sname*: text, *rating*: integer, *age*: real)

Reserves (*sid*: integer, *bid*: integer, *day*: date, *rname*: text)

- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
  - Assume there are 100 boats (each equally likely)
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
  - Assume there are 10 different ratings (each equally likely)
- Assume we have B = 5 pages to use for joins
- Remember: just counting IOs

Motivating Example: Plan 1

- Here's a reasonable query plan:

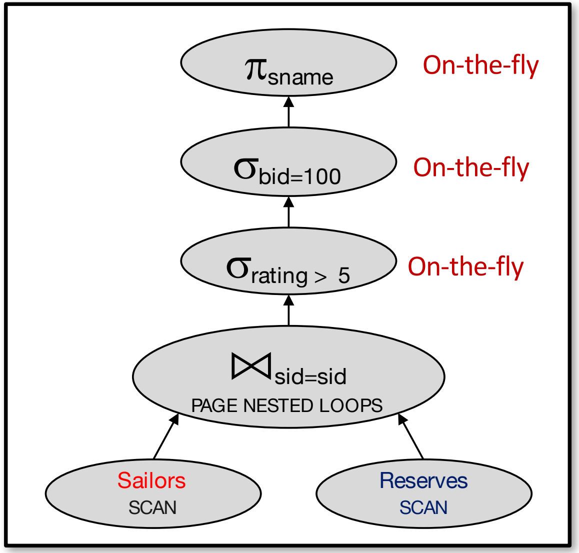


```
SELECT S.sname
  FROM Reserves R, Sailors S
 WHERE R.sid=S.sid
   AND R.bid=100
   AND S.rating>5
```

- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
  - Assume there are 100 boats (each equally likely)
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
  - Assume there are 10 different ratings (each equally likely)

- Assume we have B = 5 pages to use for joins

Motivating Example: Plan 1 Cost

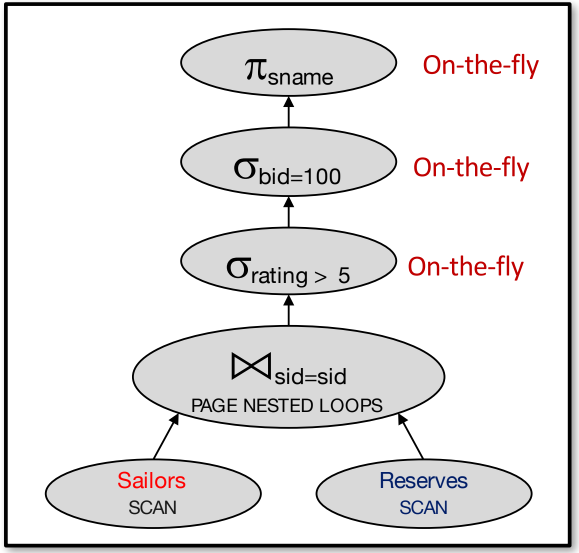


- Let's estimate the cost:
- Scan Sailors (500 IOs)
- For each page of Sailors,
  Scan Reserves (1000 IOs)
- Total: 500 + 500*1000
  - 500,500 IOs

- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
  - Assume there are 100 boats (each equally likely)
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
  - Assume there are 10 different ratings (each equally likely)

- Assume we have B = 5 pages to use for joins

Motivating Example: Plan 1 Cost Analysis

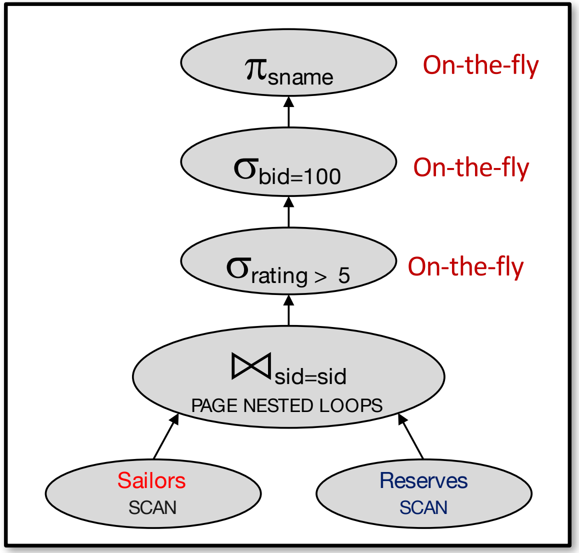


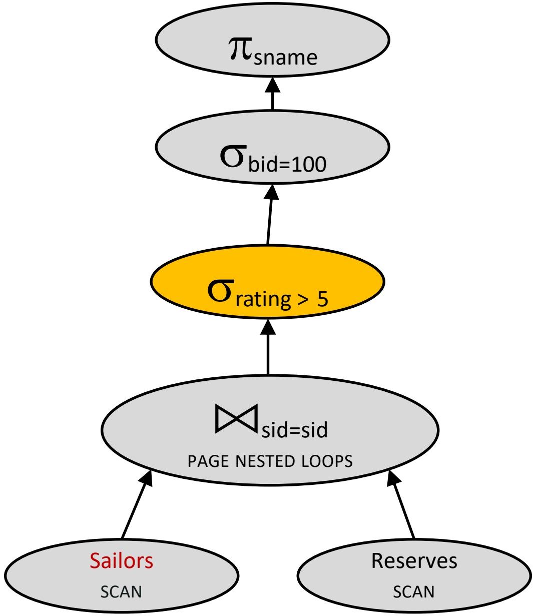
- Cost: 500+500*1000 I/Os
- By no means the worst plan!
- Misses several opportunities:
  - selections could be ‘pushed’ down
  - no use of indexes
- Goal of optimization:
  - Find faster plans that compute the same answer.

- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
  - Assume there are 100 boats (each equally likely)
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
  - Assume there are 10 different ratings (each equally likely)

- Assume we have B = 5 pages to use for joins

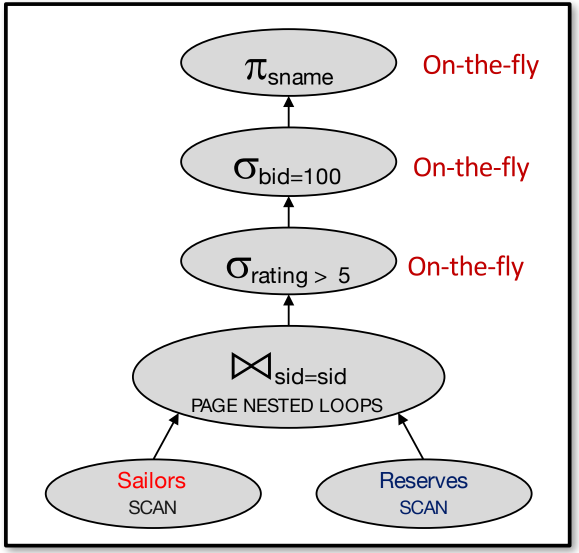
Selection Pushdown

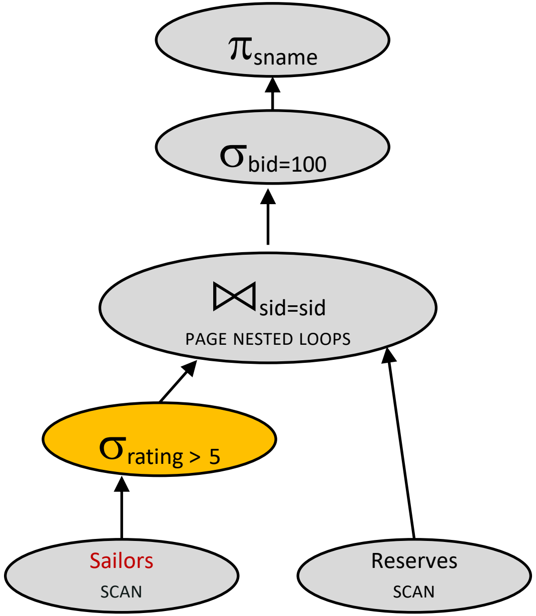




500,500 IOs

Selection Pushdown, cont



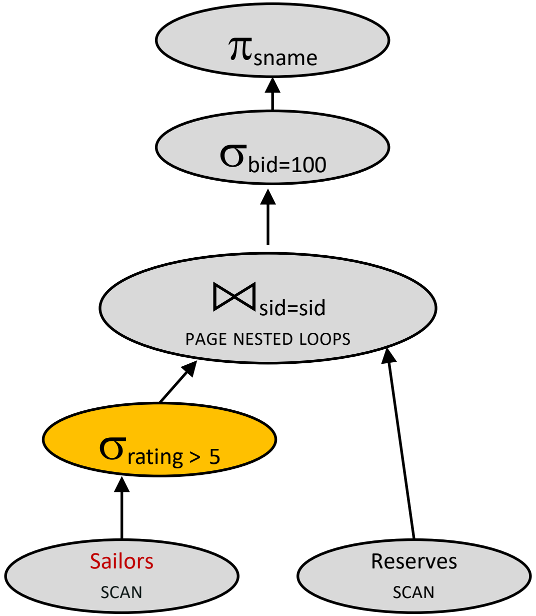


500,500 IOs

Cost?

Query Plan 2 Cost

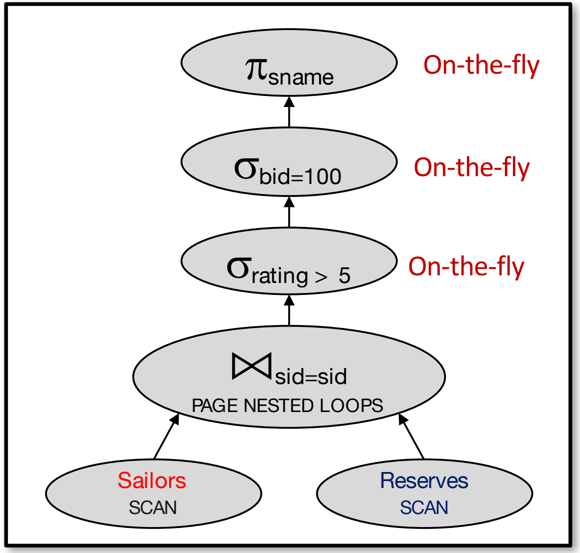
- Let's estimate the cost:
- Scan Sailors (500 IOs)
- For each pageful of high-rated Sailors,
  Scan Reserves (1000 IOs)
- Total: 500 + ???*1000
- Remember: 10 ratings, all equally likely
- Total: 500 + 250*1000



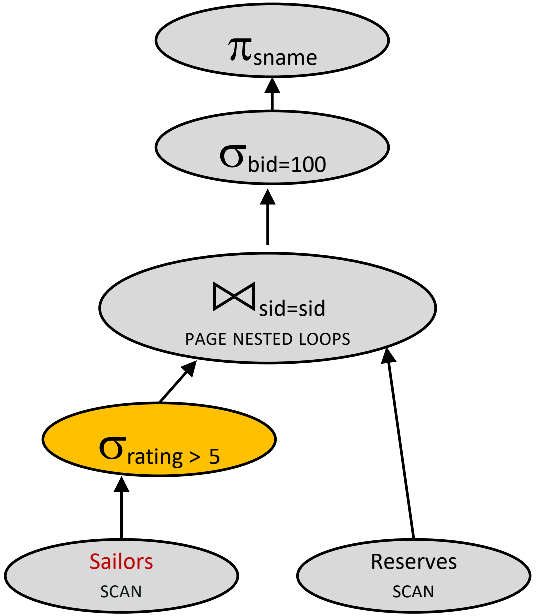
- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
  - Assume there are 100 boats (each equally likely)
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
  - Assume there are 10 different ratings (each equally likely)

- Assume we have B = 5 pages to use for joins

Decision?

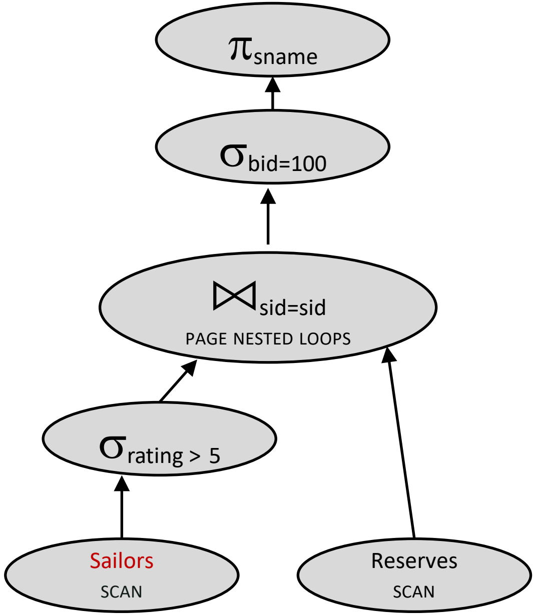


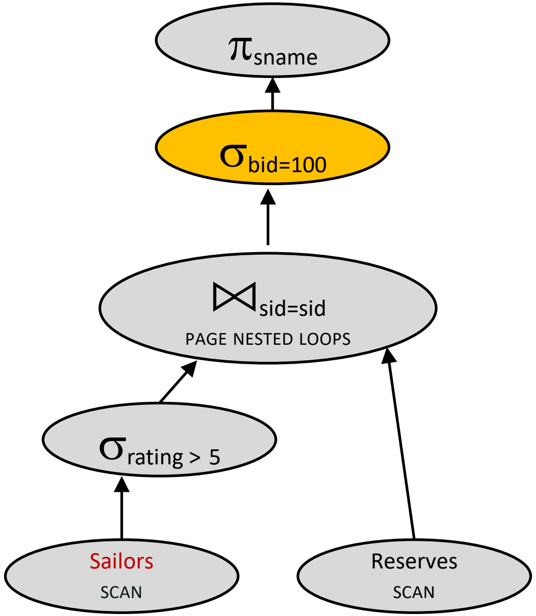
500,500 IOs



250,500 IOs

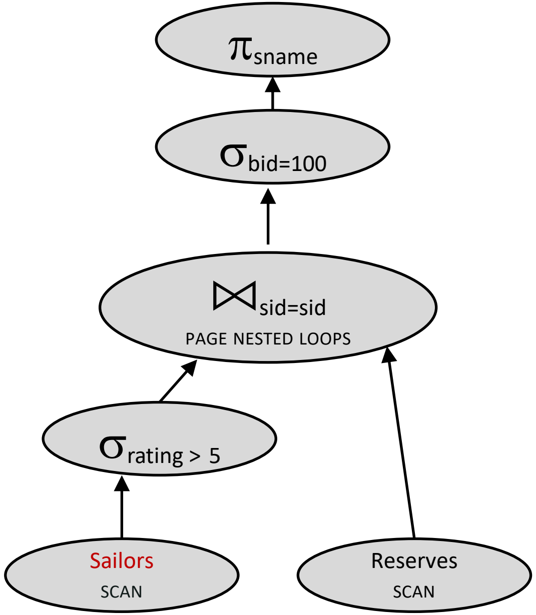
More Selection Pushdown



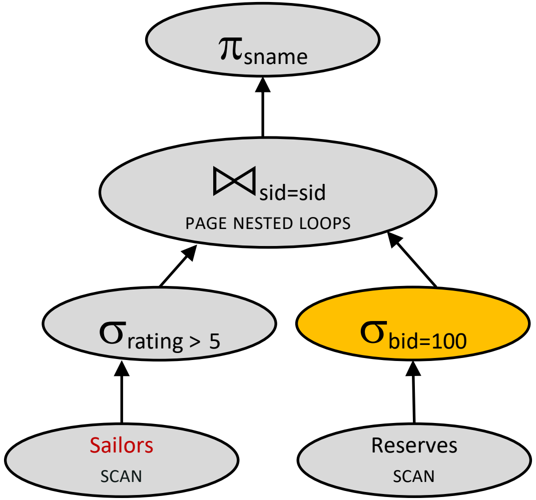


250,500 IOs

More Selection Pushdown, cont



250,500 IOs



Cost???

Query Plan 3 Cost Analysis

Let's estimate the cost:

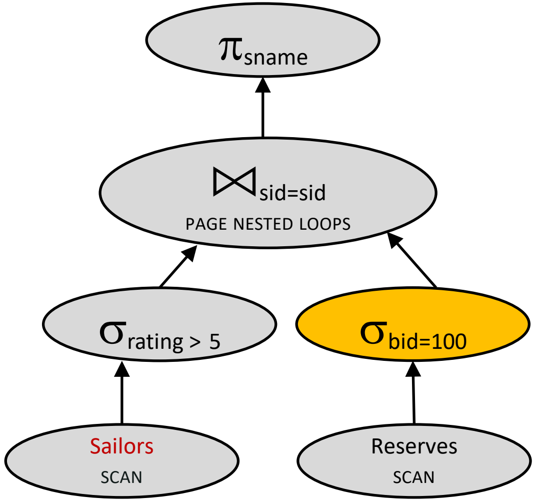
- Scan Sailors (500 IOs)
- For each pageful of high-rated Sailors,
  Read through Reserves tuples that match

- Total: 500 + 250*(???)

- For each scan of Reserves, we apply filter on tuples on the fly

- Problem: this doesn't actually save any IOs – to determine the Reserves tuples that match, we end up scanning Reserves the same # of times.

- Total: 500 + 250*1000!

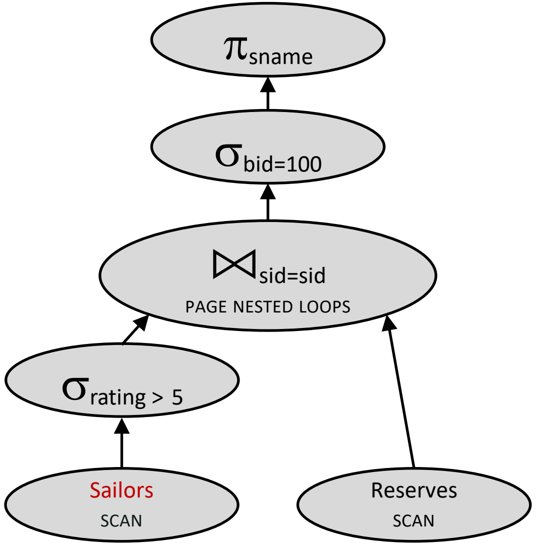


- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
  - Assume there are 100 boats (each equally likely)
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
  - Assume there are 10 different ratings (each equally likely)

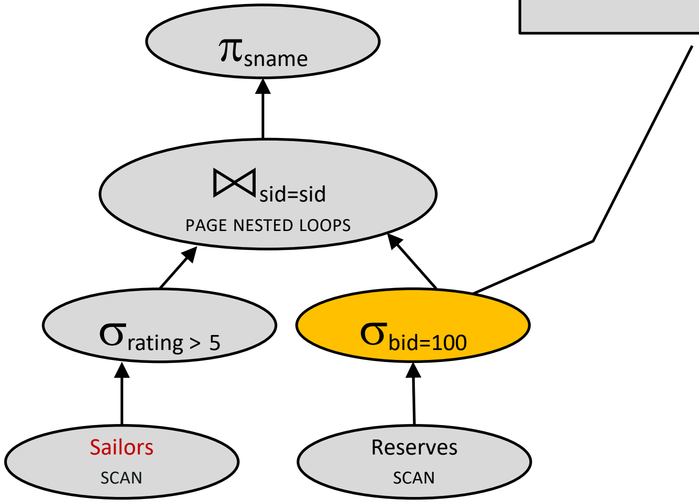
- Assume we have B = 5 pages to use for joins

More Selection Pushdown Analysis

Pushing a selection into the inner loop of a nested loop join doesn't save I/Os! Essentially equivalent to having the selection above.

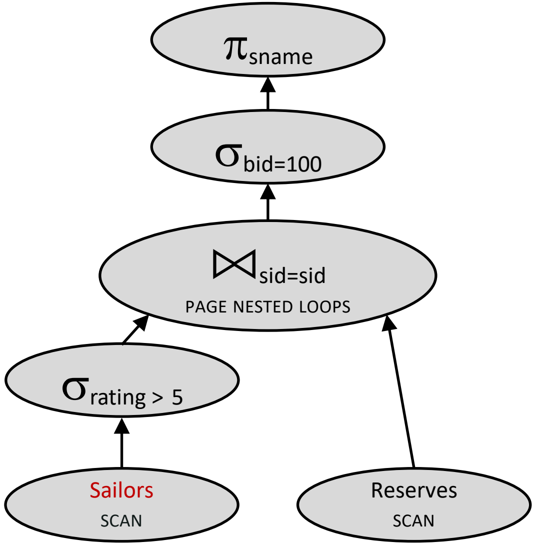


250,500 IOs

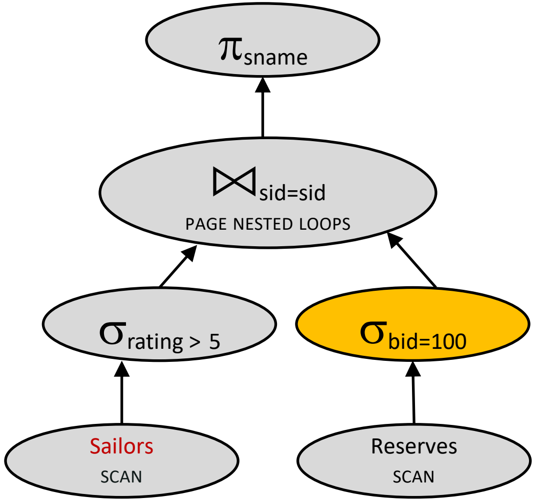


250,500 IOs

Decision 2



250,500 IOs



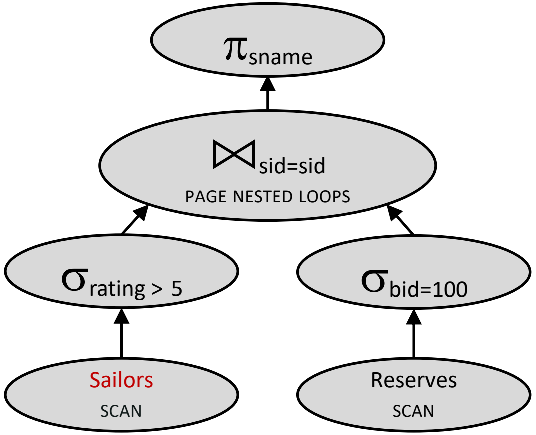
250,500 IOs

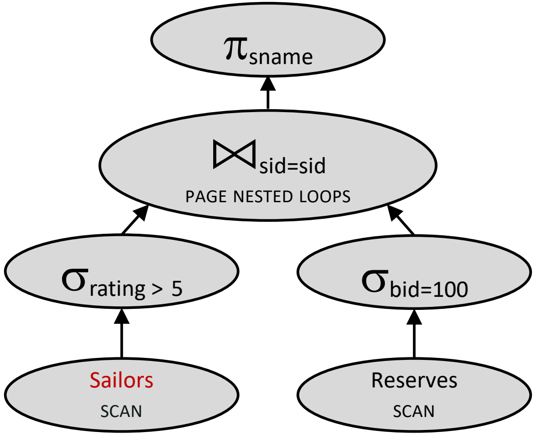
So far, we’ve tried

- Basic page nested loops (500,500)
- Selection pushdown on left (250,500)
- More selection pushdown on right (250,500)

- Next up, join ordering

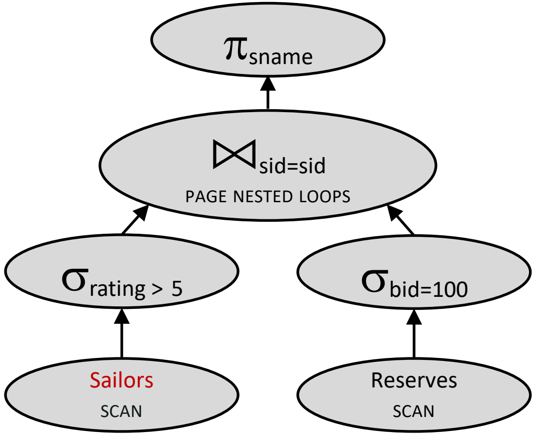
Next up: Join Ordering

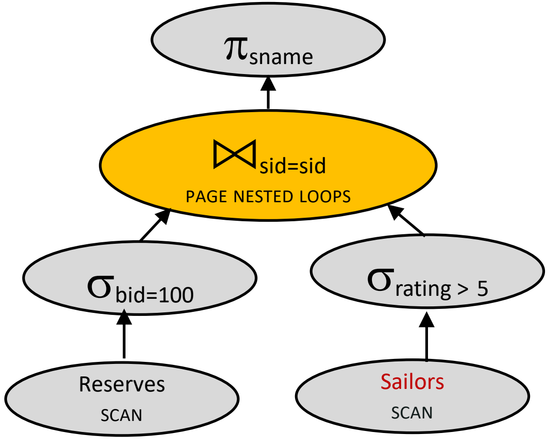




250,500 IOs

Join Ordering, cont

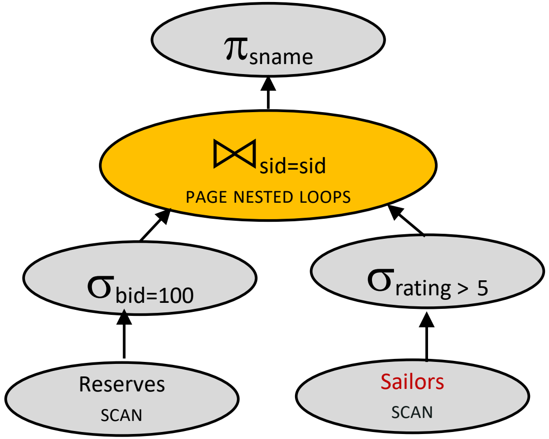




250,500 IOs

Query Plan 4 Cost

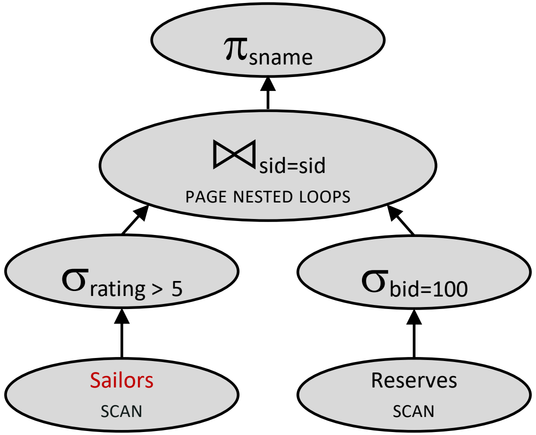
- Let's estimate the cost:
- Scan Reserves (1000 IOs)
- For each pageful of Reserves for bid 100,
  Scan Sailors (500 IOs)
- Total: 1000 +???*500
- Uniformly distributed across 100 boat values
- Total: 1000 +10*500



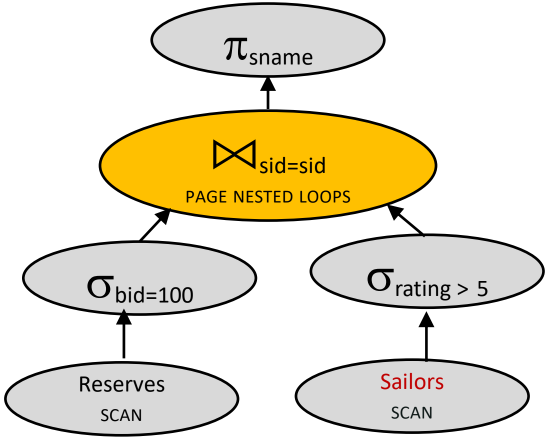
- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
  - Assume there are 100 boats (each equally likely)
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
  - Assume there are 10 different ratings (each equally likely)

- Assume we have B = 5 pages to use for joins

Decision 3



250,500 IOs



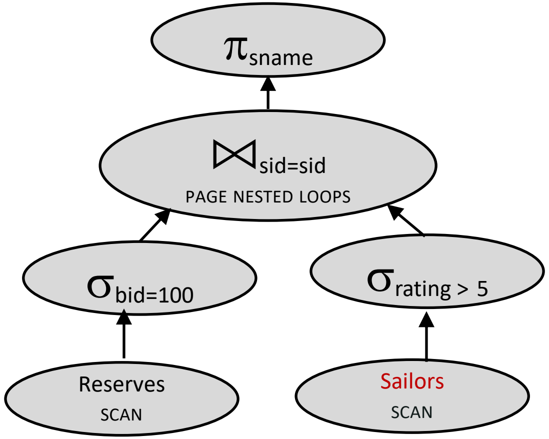
6000 IOs

So far, we’ve tried

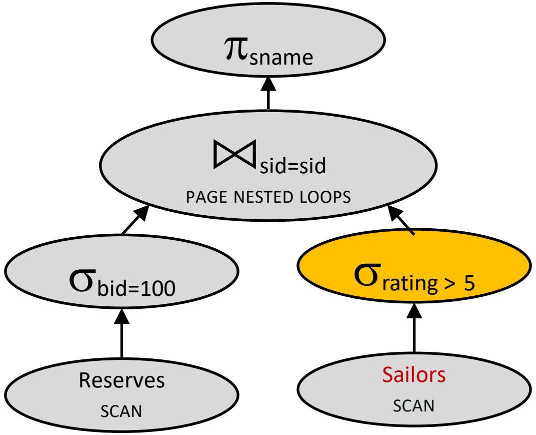
- Basic page nested loops (500,500)
- Selection pushdown on left (250,500)
- More selection pushdown on right (250,500)
- Join ordering (6000)

- Next up, materialization …

Materializing Inner Loops



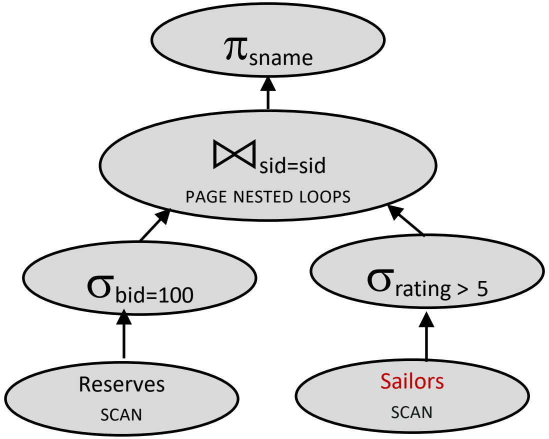
6000 IOs



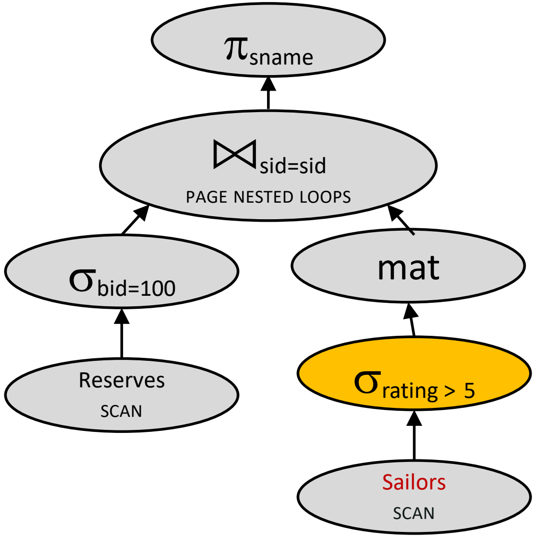
If you recall, selection pushdown on the right doesn't help because it is done on the fly.

What if we materialize the result after the selection?

Materializing Inner Loops, cont



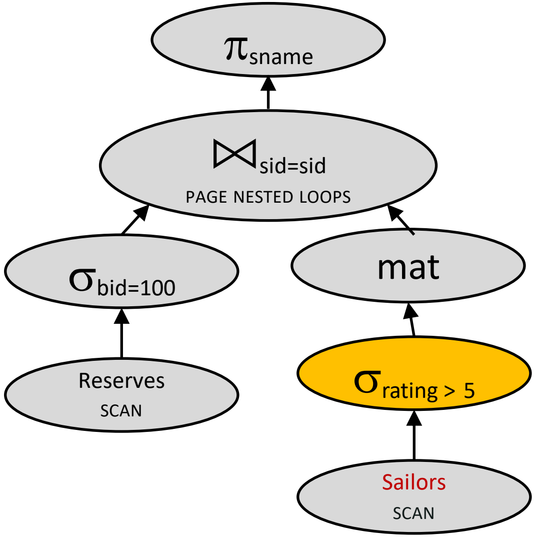
6000 IOs



Cost???

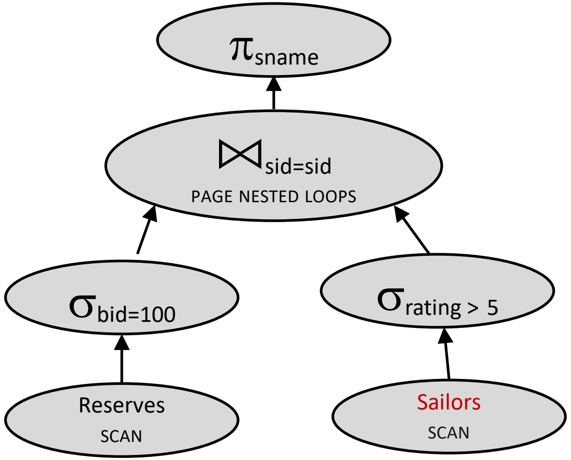
Plan 5 Cost Analysis

- Let's estimate the cost:
- Scan Reserves (1000 IOs)
- Scan Sailors (500 IOs)
- Materialize Temp table T1 (??? IOs)
- For each pageful of Reserves for bid 100,
  Scan T1 (??? IOs)
- Total: 1000 + 500 + ??? + 10*???
- 1000 + 500+ 250 + (10 * 250)

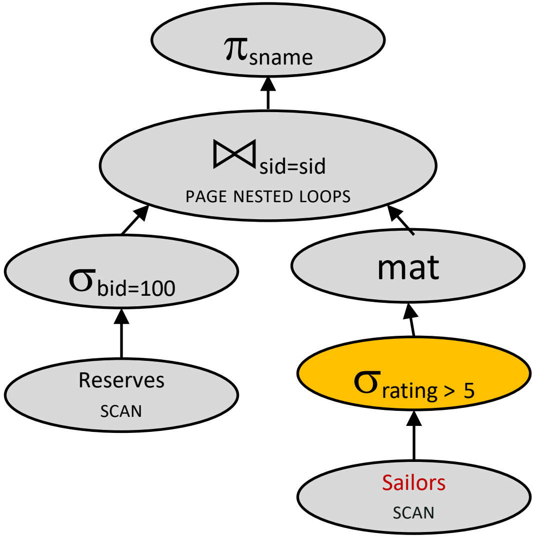


- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
  - Assume there are 100 boats (each equally likely)
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
  - Assume there are 10 different ratings (each equally likely)
- Assume we have B = 5 pages to use for joins

Materializing Inner Loops, cont.

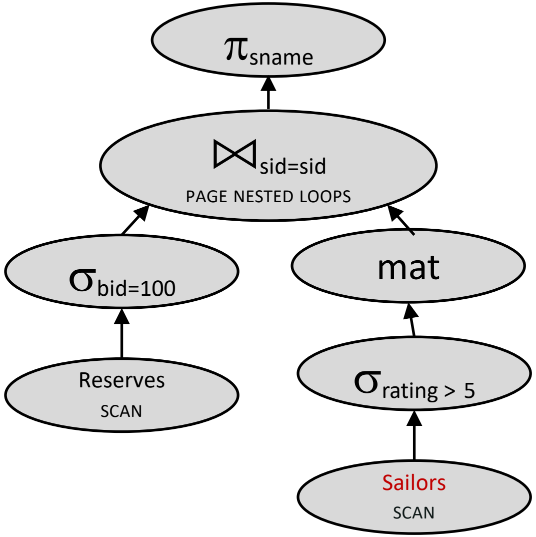


6000 IOs

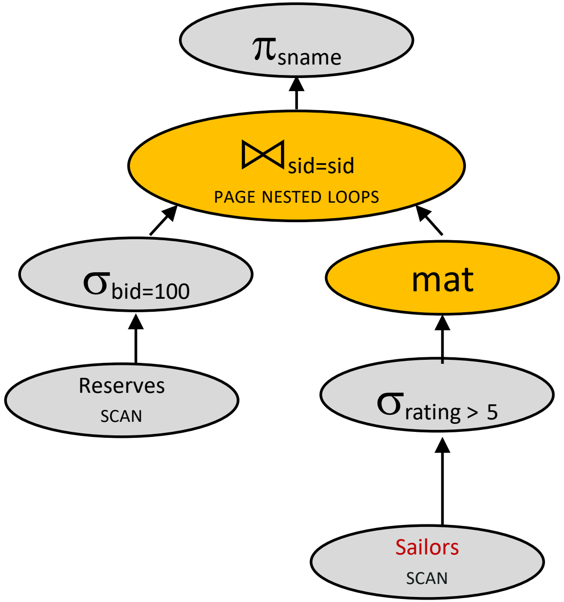


4250 IOs

Join Ordering Again

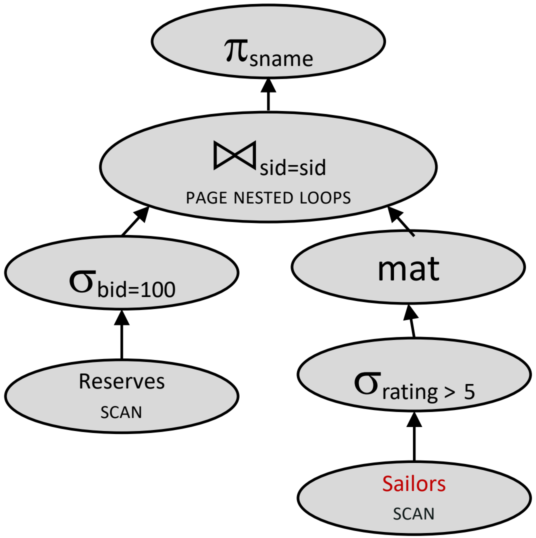


4250 IOs



Let's try flipping the join order again with materialization trick

Join Ordering Again, Cont



π_{sname}

$\bowtie_{sid=sid}$

PAGE NESTED LOOPS

$\sigma_{bid=100}$

Reserves

SCAN

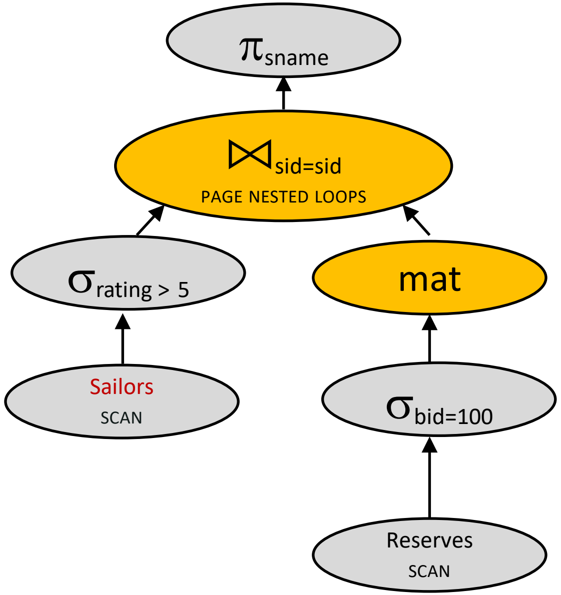
mat

$\sigma_{rating > 5}$

Sailors

SCAN

4250 IOs



π_{sname}

$\bowtie_{sid=sid}$

PAGE NESTED LOOPS

$\sigma_{rating > 5}$

Sailors

SCAN

mat

$\sigma_{bid=100}$

Reserves

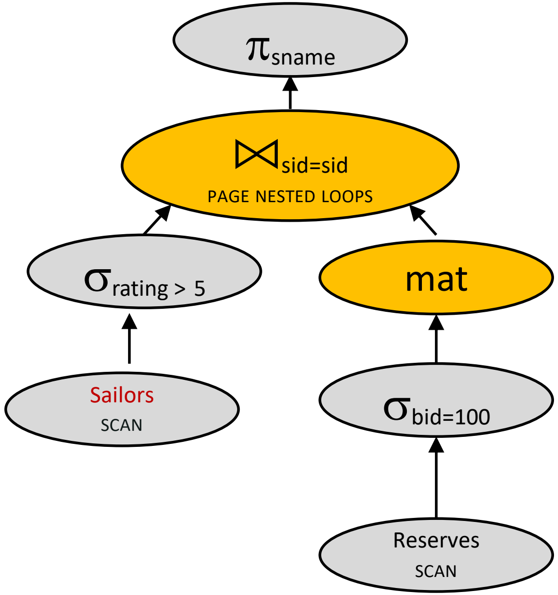
SCAN

Cost???

Let's try flipping the join order again with materialization trick

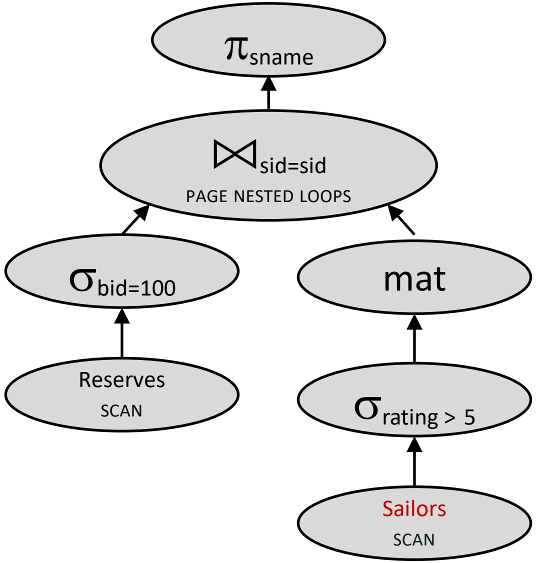
Plan 6 Cost Analysis

- Let’s estimate the cost:
- Scan Sailors (500 IOs)
- Scan Reserves (1000 IOs)
- Materialize Temp table T1 (??? IOs)
- For each pageful of high-rated Sailors,
  Scan T1 (??? IOs)
- Total: 500 + 1000 + ??? + 250*???
- 500 + 1000 +10 +(250 *10)

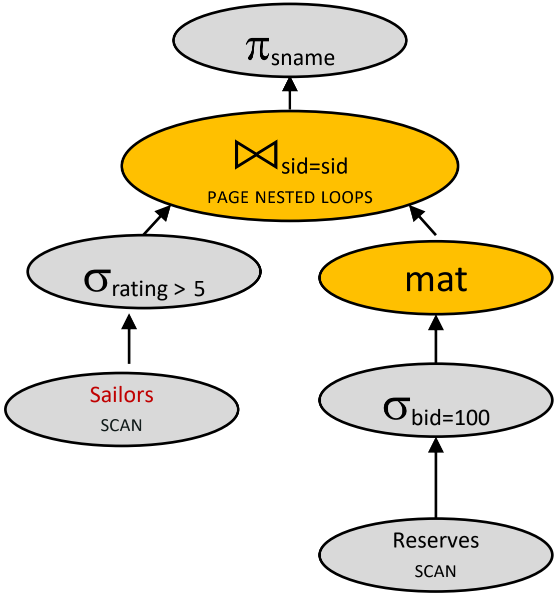


- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
  - Assume there are 100 boats (each equally likely)
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
  - Assume there are 10 different ratings (each equally likely)
- Assume we have B = 5 pages to use for joins

Decision 4



4250 IOs



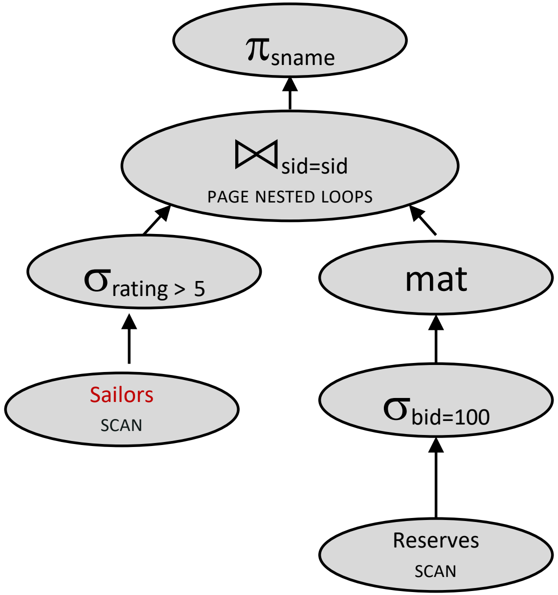
4010 IOs

So far, we’ve tried

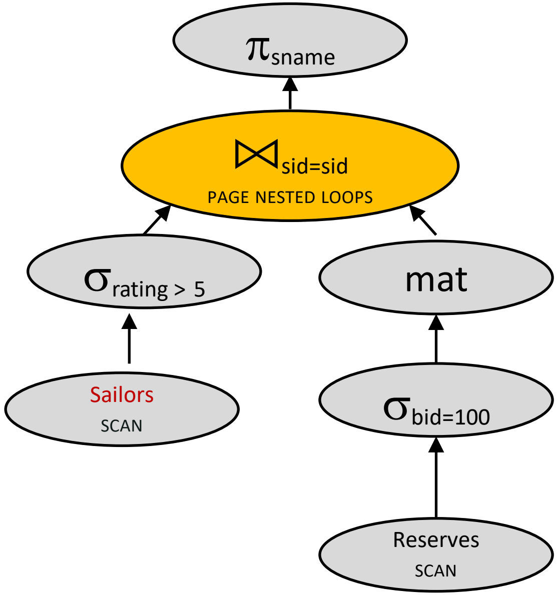
- Basic page nested loops (500,500)
- Selection pushdown on left (250,500)
- More selection pushdown on right (250,500)
- Join ordering (6000)
- Materializing inner loop (4250)
- Join ordering again with materialization (4010)

- Next up, sort merge …

Join Algorithm

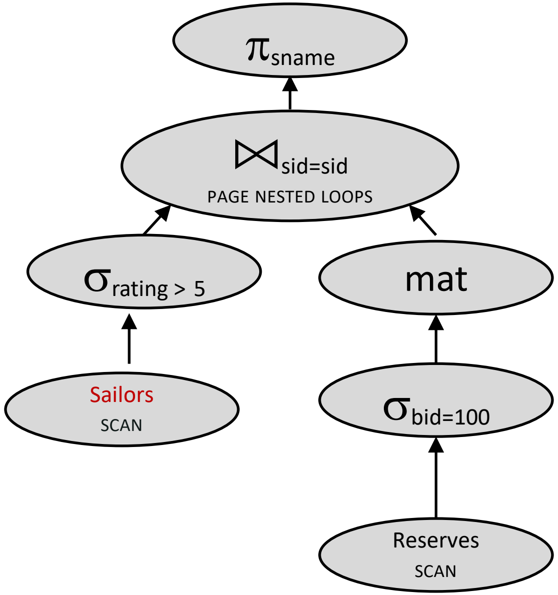


4010 IOs

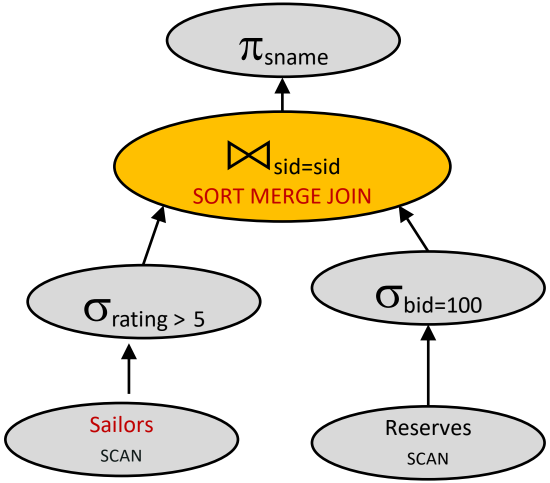


What if we change the join algorithm?

Join Algorithm, cont.



4010 IOs

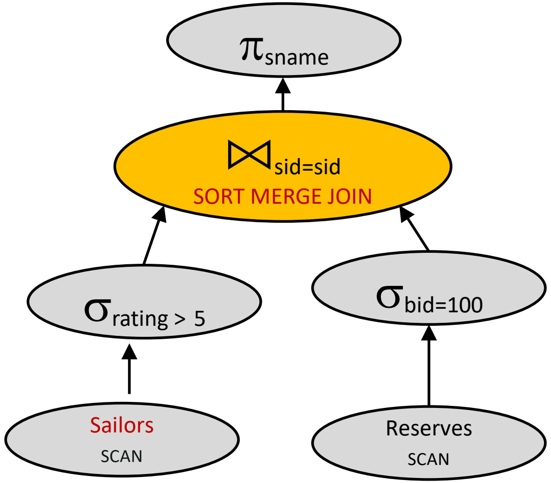


Cost???

What if we change the join algorithm?

Query Plan 7 Cost Analysis

- With 5 buffers, cost of plan:
- Scan Reserves (1000)
- Scan Sailors (500)
- Sort high-rated sailors (???)
  Note: pass 0 doesn't do read I/O, just gets input from select.
- Sort reservations for boat 100 (???)
  Note: pass 0 doesn't do read I/O, just gets input from select.
- How many passes for each sort?
- Merge (10+250) = 260
- Total:



- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
  - Assume there are 100 boats (each equally likely)
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
  - Assume there are 10 different ratings (each equally likely)
- Assume we have B = 5 pages to use for joins

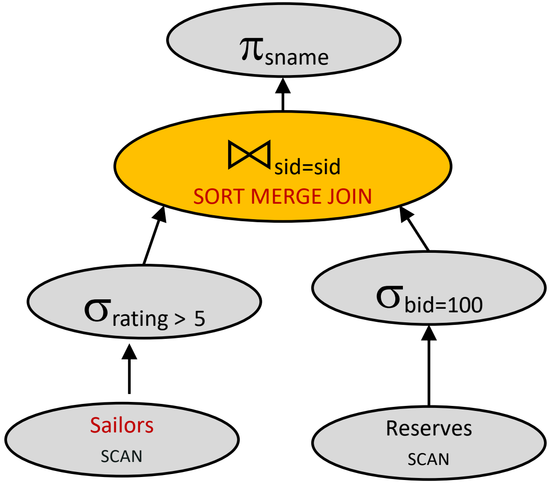
Query Plan 7 Cost Analysis Part 2

- With 5 buffers, cost of plan:
- Scan Reserves (1000)
- Scan Sailors (500)

- Sort
  - 2 passes for reserves
    pass 0 = 10 to write, pass 1 = 2*10 to read/write
  - 4 passes for sailors
    pass 0 = 250 to write, pass 1,2,3 = 2*250 to read/write

- Merge (10+250) = 260

Scan both (1000 + 500) + sort reserves(10 + 2*10) + sort sailors (250 + 3*2*250) + merge (10+250) = 3540



- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
  - Assume there are 100 boats (each equally likely)
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
  - Assume there are 10 different ratings (each equally likely)

- Assume we have B = 5 pages to use for joins

Query Plan 7 Cost Analysis Part 2

- With 5 buffers, cost of plan:
- Scan Reserves (1000)
- Scan Sailors (500)

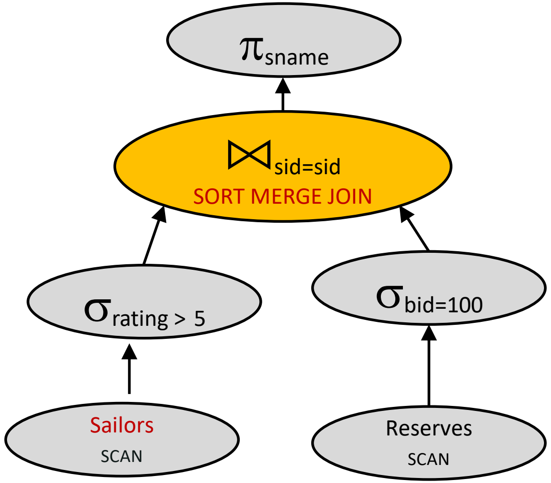
- Sort

  - 2 passes for reserves
    pass 0 = 10 to write (2 runs of 5 each); pass 1 = 2*10 to read/write (one representative from 2 runs)

  - 4 passes for sailors
    pass 0 = 250 to write (50 runs of 5 each); pass 1 (merging to give 50/4=13 runs of 4 * 5 size each); pass 2 (merging to give 13/4=4 runs of 4 * 4 * 5 size each); pass 3 (merging to give one run of 250 in total) pass 1,2,3 = 2*250 to read/write

- Merge (10+250) = 260

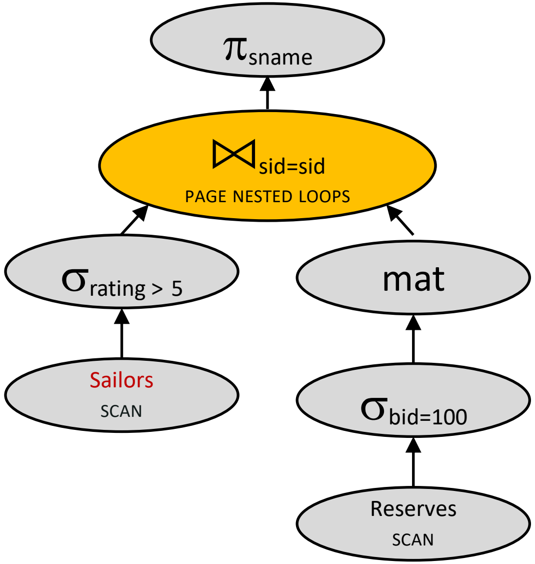
Scan both (1000 + 500) + sort reserves(10 + 2*10) + sort sailors (250 + 3*2*250) + merge (10+250) = 3540



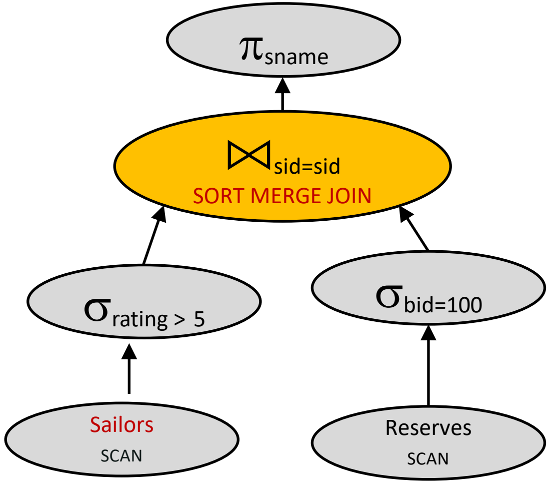
- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
  - Assume there are 100 boats (each equally likely)
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
  - Assume there are 10 different ratings (each equally likely)

- Assume we have B = 5 pages to use for joins

Decision 5



4010 IOs



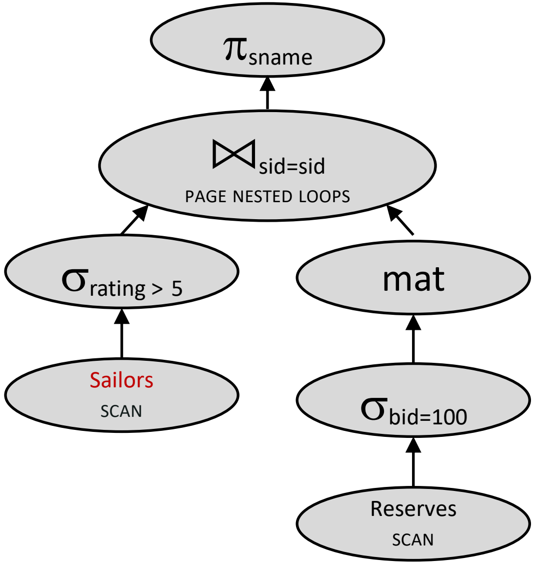
3540 IOs

So far, we’ve tried

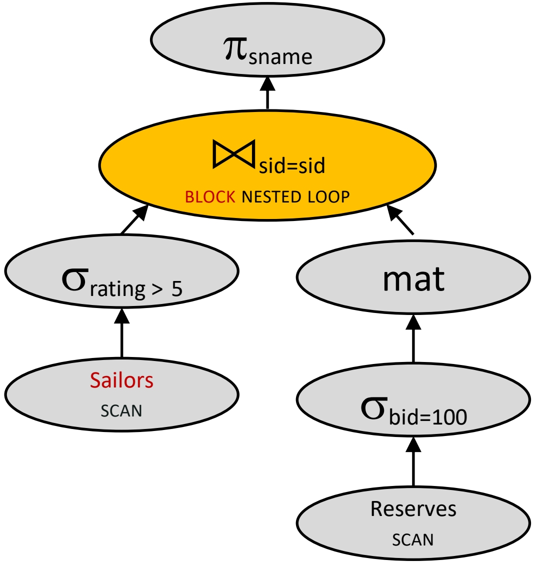
- Basic page nested loops (500,500)
- Selection pushdown on left (250,500)
- More selection pushdown on right (250,500)
- Join ordering (6000)
- Materializing inner loop (4250)
- Join ordering again with materialization (4010)
- Sort-merge join (3540)

- Next up, block nested …

Join Algorithm Again, Again



4010 IOs
(And Sort-Merge at 3540 IOs)



Cost???

Returning to our best (so far) page nested loops plan again…

Query 8 Cost Analysis

- With 5 buffers, cost of plan:

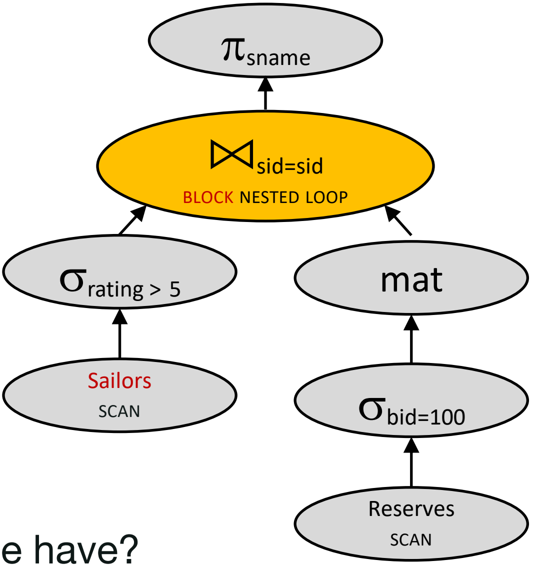
- Scan Sailors (500)
- Scan Reserves (1000)

- Write Temp T1 (10)
- For each blockful of high-rated sailors
- Loop on T1 (??? * 10)
  - What is the chunk size? How many chunks (???) will we have?
  - 3 pages; ceil(250/3)

- Total:

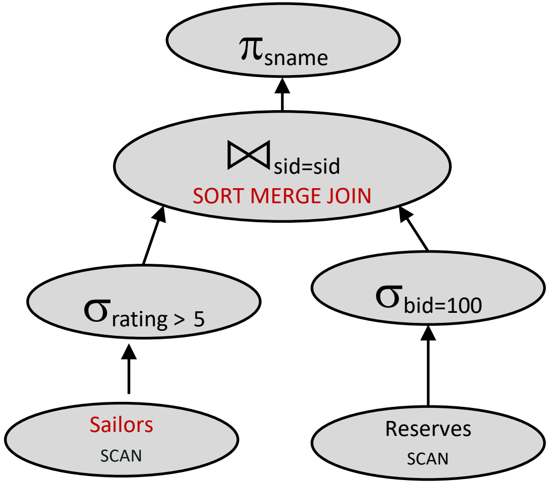
- Scan both(500 + 1000) + write out T1(10) + BNLJ (ceil(250/3) *10)

  = 500 + 1000 +10 +(84 *10) = 2350

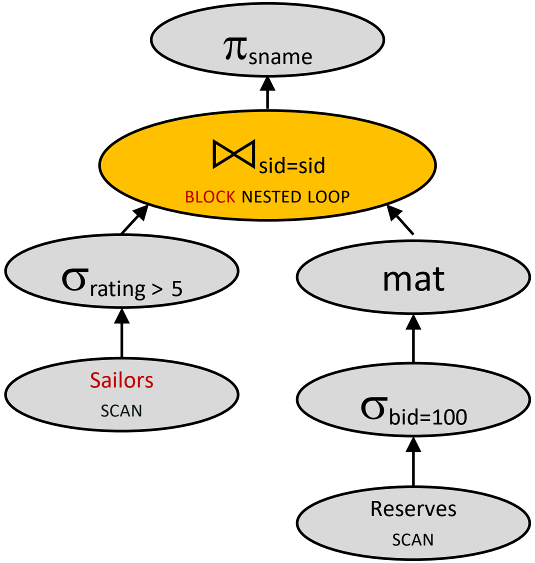


- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
  - Assume there are 100 boats (each equally likely)
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
  - Assume there are 10 different ratings (each equally likely)

Decision 6



3540 IOs



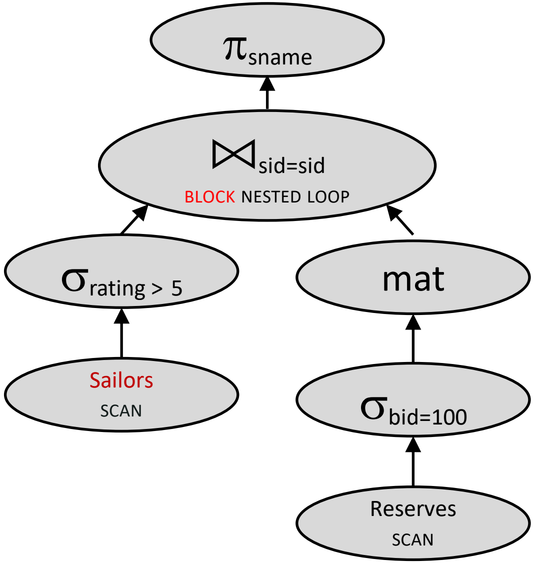
2350 IOs

So far, we’ve tried

- Basic page nested loops (500,500)
- Selection pushdown on left (250,500)
- More selection pushdown on right (250,500)
- Join ordering (6000)
- Materializing inner loop (4250)
- Join ordering again with materialization (4010)
- Sort-merge join (3540)
- Block nested loops (2350)

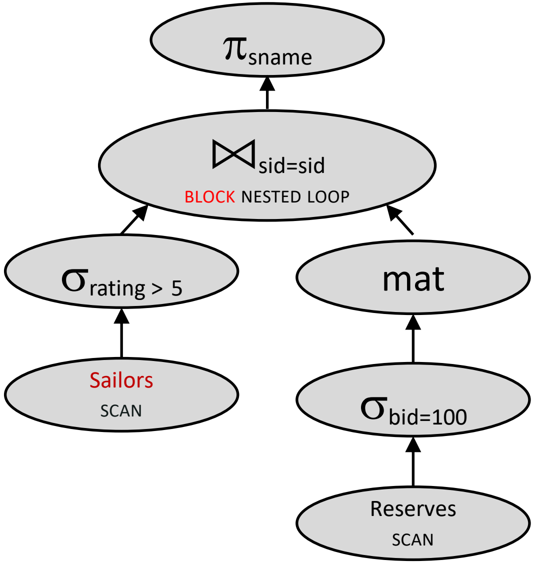
- Next up, projection cascade

Projection Cascade & Pushdown

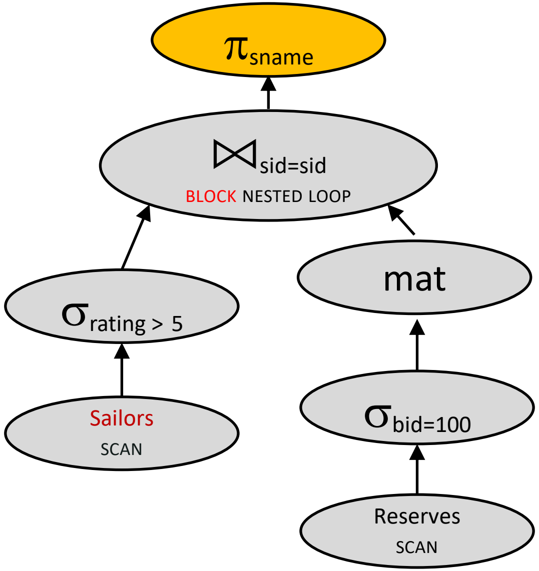


2350 IOs

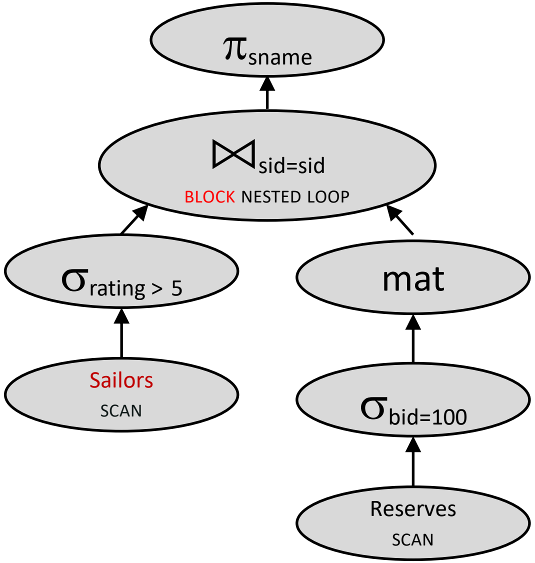
Projection Cascade & Pushdown, cont



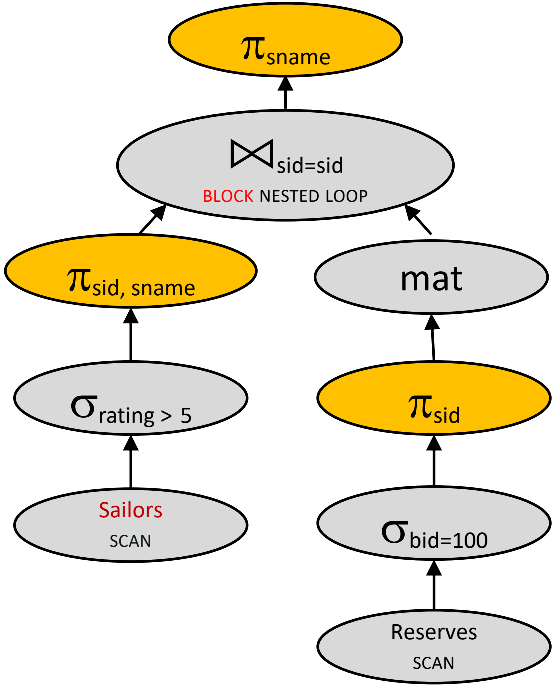
2350 IOs



Projection Cascade & Pushdown, cont



2350 IOs

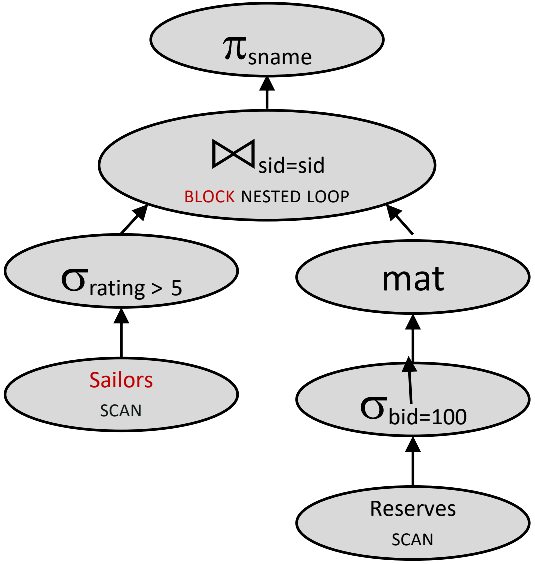


Super small!
Single page – can make this the “chunk” on the left

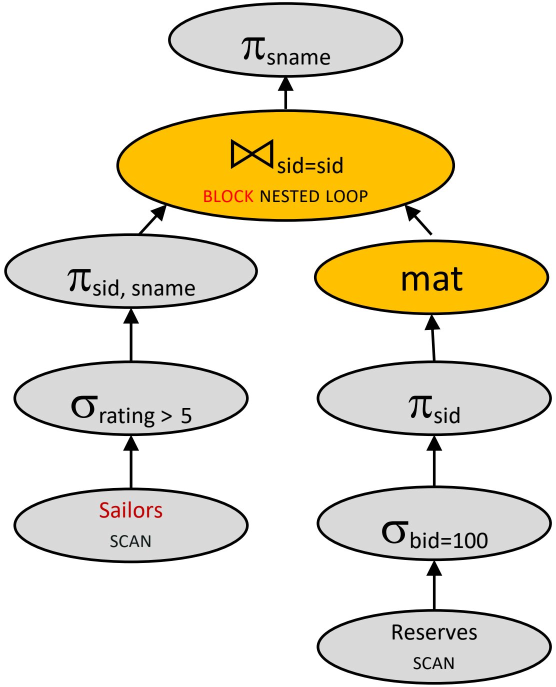
1 page [4 out of 40 bytes

10 pages

With Join Reordering, no Mat



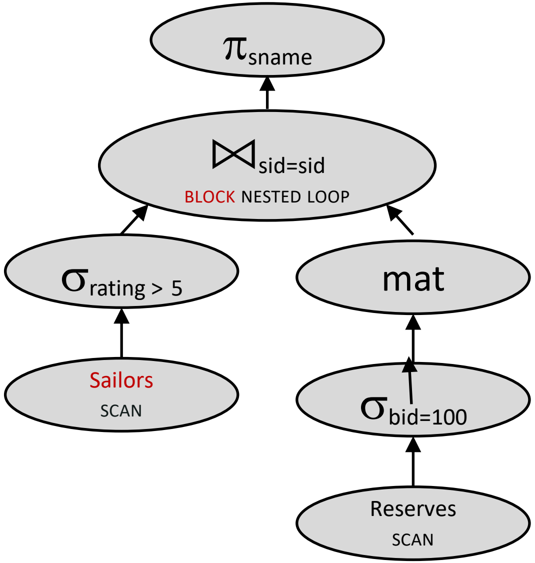
2350 IOs



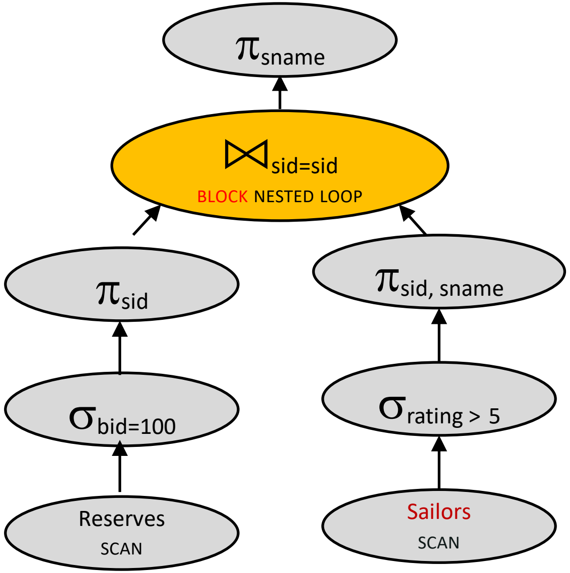
So we'll try reordering the join again.

We'll also skip on the materialization for this (convince yourself later that it doesn't help)

With Join Reordering, no Mat cont



2350 IOs

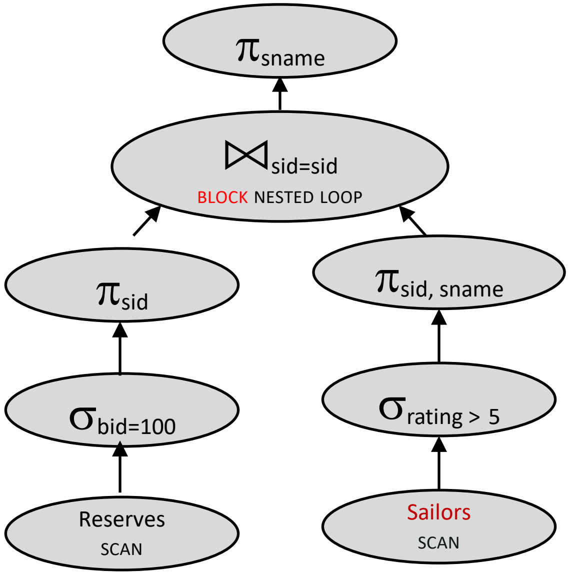


So we'll try reordering the join again.

We'll also skip on the materialization for this (convince yourself later that it doesn't help)

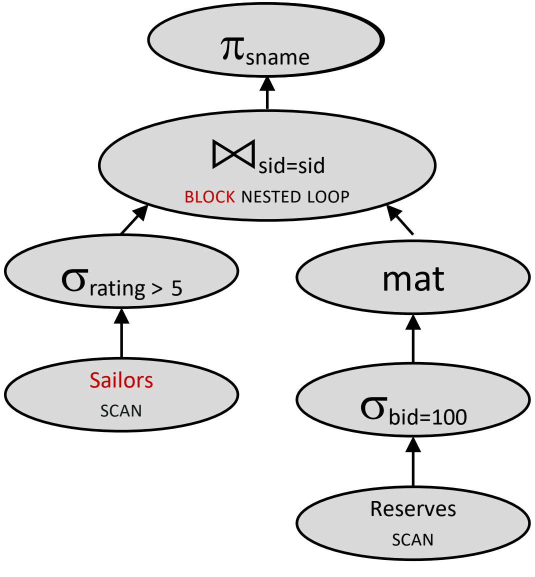
Query Plan 9 Cost Analysis

- With 5 buffers, cost of plan:
- Scan Reserves (1000)
- For each blockful of sids that rented boat 100
- (recall Reserve tuple is 40 bytes, assume sid is 4 bytes)
  - 10 pages down to 1 page
- Loop on Sailors (??? * 500) = 1 * 500
- Total: 1500

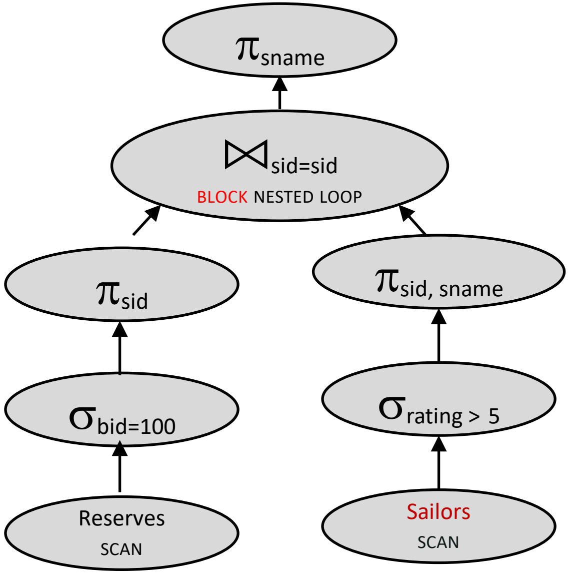


- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
  - Assume there are 100 boats (each equally likely)
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
  - Assume there are 10 different ratings (each equally likely)
- Assume we have B = 5 pages to use for joins

With Join Reordering, no Mat, cont.



2350 IOs



1500 IOs <= Can't do much better w/o indexes! Why?

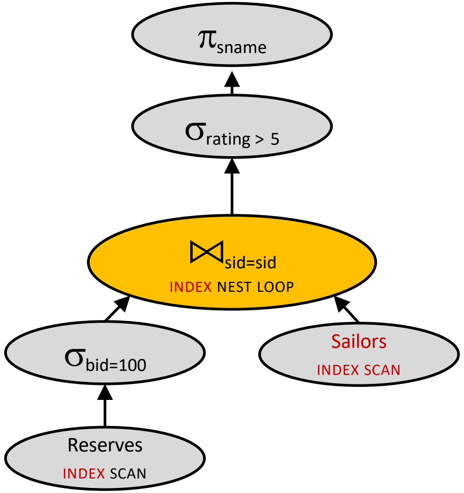
So far, we’ve tried

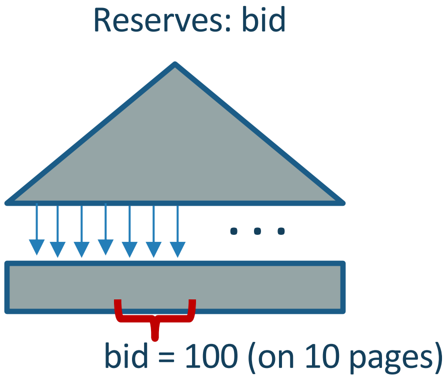
- Basic page nested loops (500,500)
- Selection pushdown on left (250,500)
- More selection pushdown on right (250,500)
- Join ordering (6000)
- Materializing inner loop (4250)
- Join ordering again with materialization (4010)
- Sort-merge join (3540)
- Block nested loops (2350)
- Projection cascade, plus reordering again (1500)

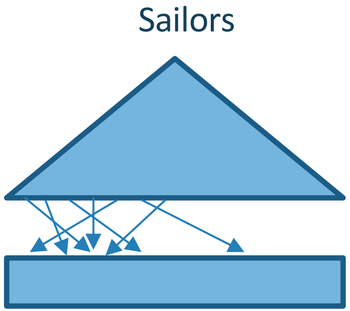
- Next up, indexes

How About Indexes?

- Indexes:
  - Reserves.bid clustered
  - Sailors.sid unclustered
- Assume indexes fit in memory





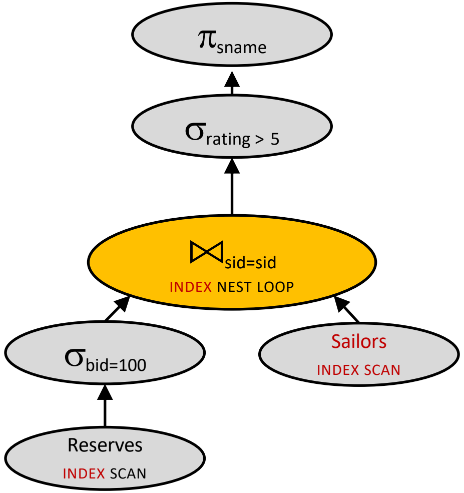


- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
  - Assume there are 100 boats (each equally likely)
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
  - Assume there are 10 different ratings (each equally likely)
- Assume we have B = 5 pages to use for joins

Index Cost Analysis

In our query plan, note:

- **No projection pushdown to left** for π_{sname}
  - Projecting out unnecessary fields from outer of Index NL doesn't make an I/O difference (still doing things per tuple)
- **No selection pushdown to right** for $\sigma_{\text{rating} > 5}$
  - Does not affect Sailors.sid index lookup
- With clustered index on bid of Reserves, we access how many pages of Reserves?:
  - 100,000/100 = 1000 tuples on 1000/100 = 10 pages.
- Join column sid is a **key** for Sailors.
  - At most one matching tuple using unclustered index on sid



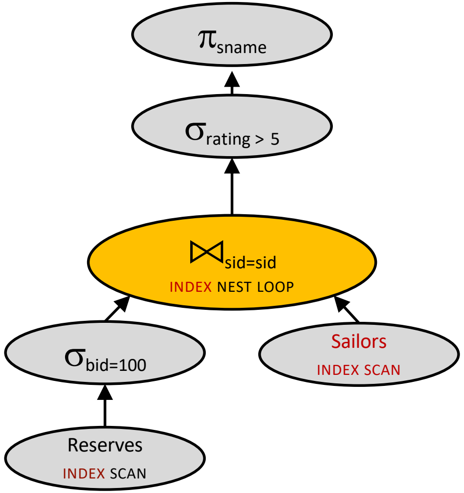
π_{sname}
$\sigma_{\text{rating > 5}}$
$\bowtie_{\text{sid=sid}}$
INDEX NEST LOOP
$\sigma_{\text{bid=100}}$
Sailors
INDEX SCAN
Reserves
INDEX SCAN

1010 IOs

- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
  - Assume there are 100 boats (each equally likely)
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
  - Assume there are 10 different ratings (each equally likely)
- Assume we have B = 5 pages to use for joins

Index Cost Analysis Part 2

- With clustered index on bid of Reserves, we access how many pages of Reserves?:
  - 100,000/100 = 1000 tuples on 1000/100 = 10 pages.
- for each Reserves tuple (1000 such tuples)
  get matching Sailors tuple (1 IO)
- 10 + 1000*1
- Cost: Selection of Reserves tuples (10 I/Os); then, for each, must get matching Sailors tuple (1000); total 1010 I/Os.



1010 IOs

- Reserves:
  - Each tuple is 40 bytes long, 100 tuples per page, 1000 pages.
  - Assume there are 100 boats (each equally likely)
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
  - Assume there are 10 different ratings (each equally likely)
- Assume we have B = 5 pages to use for joins

The Entire Story

- Basic page nested loops (500,500)
- Selection pushdown on left (250,500)
- More selection pushdown on right (250,500)
- Join ordering (6000)
- Materializing inner loop (4250)
- Join ordering again with materialization (4010)
- Sort-merge join (3540)
- Block nested loops (2350)
- Projection cascade, plus reordering again (1500)
- Index nested loops (1010)

- Still only a subset of the possible plans for this query!!!

Summing up

- There are *lots* of plans
  - Even for a relatively simple query
- Engineers often think they can pick good ones
  - E.g. MapReduce API was based on that assumption
  - So was the COBOL API of 1970's!
- Not so clear that's true!
  - Manual query planning can be tedious, technical
  - Machines are better at enumerating options than people
  - We will see soon how optimizers make simplifying assumptions to examine a reasonable set of plans