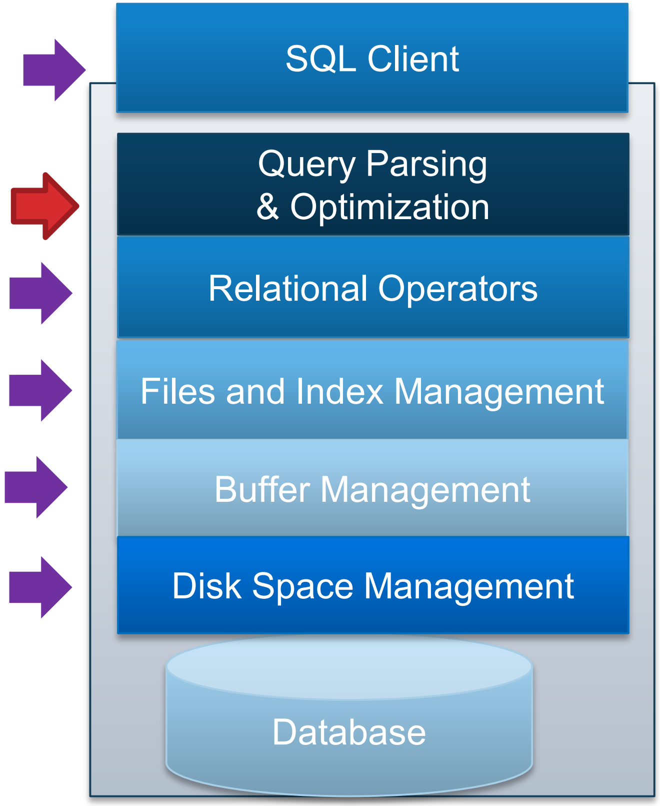
Relational Query Optimization I: The Plan Space

R&G 15

Architecture of a DBMS

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

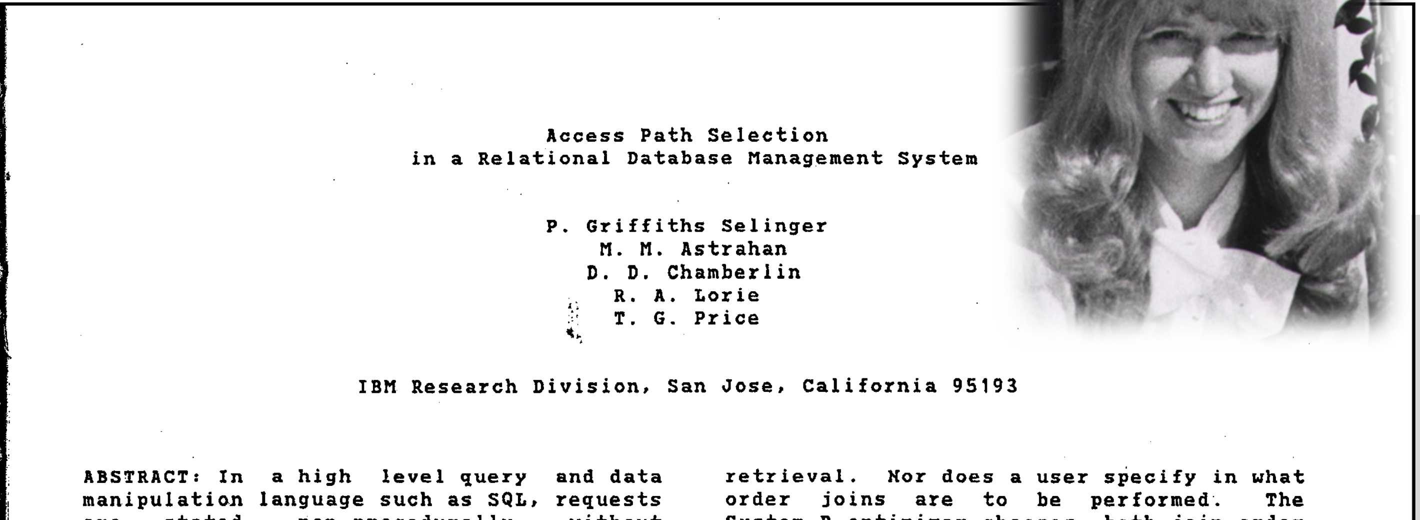


Query Optimization is Magic

- The bridge between a *declarative* domain-specific language…
  - “What” you want as an answer
- … and custom *imperative* computer programs
  - “How” to compute the answer
- In 2018 terms:
  - This is AI-driven Software Synthesis
  - That’s not just marketing!
    - Similar to cutting edge AI work today
    - Optimization + heuristic pruning
    - Research exploring the use of modern AI techniques to improve that pruning (e.g. Deep Reinforcement Learning)

Invented in 1979 by Pat Selinger et al.

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



Access Path Selection
in a Relational Database Management System

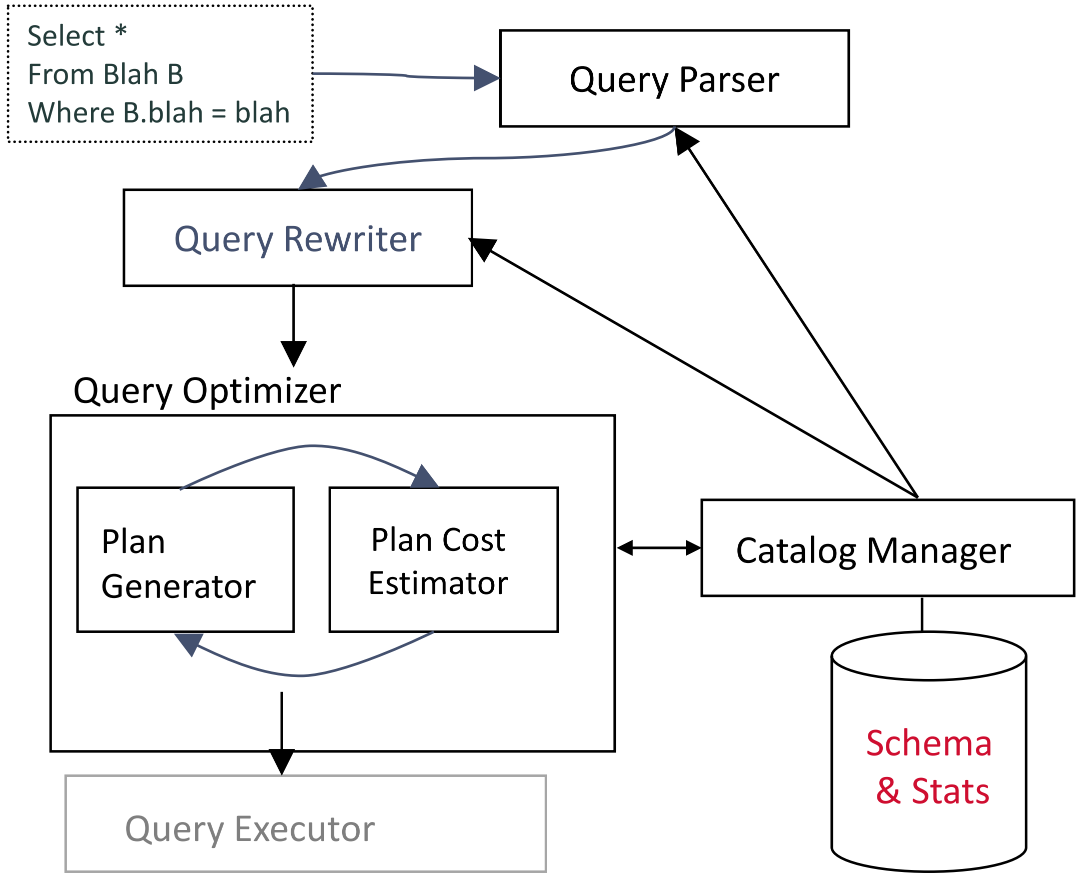
P. Griffiths Selinger
M. M. Astrahan
D. D. Chamberlin
R. A. Lorie
T. G. Price

IBM Research Division, San Jose, California 95193

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

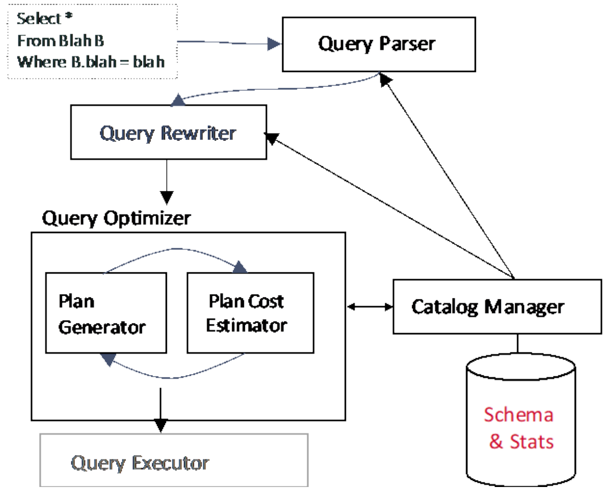
retrieval. Nor does a user specify in what order joins are to be performed. The

Query Parsing & Optimization



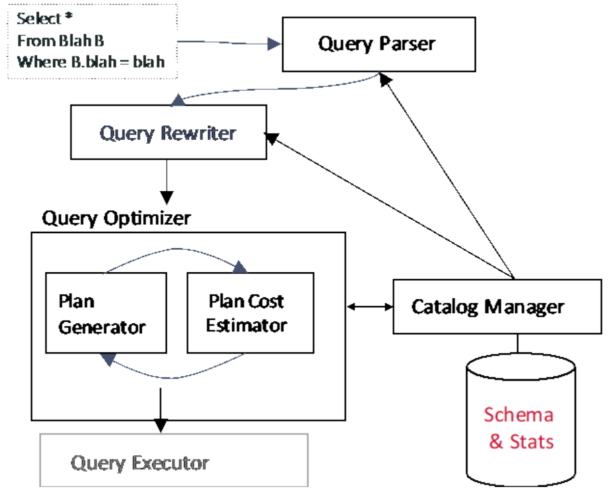
Query Parsing & Optimization Part 2

- Query parser
  - Checks correctness, authorization
  - Generates a parse tree
  - Straightforward



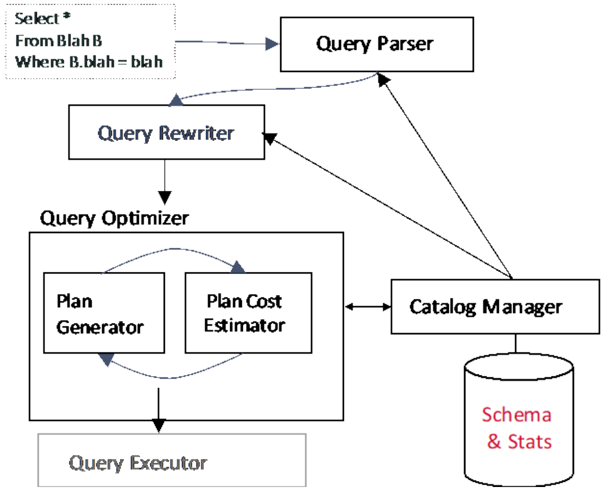
Query Parsing & Optimization Part 3

- Query rewriter
    - Converts queries to canonical form
        - flatten views
        - subqueries into fewer query blocks
    - Weak spot in many open-source DBMSs



Query Parsing & Optimization Part 4

- "Cost-based" Query Optimizer
  - 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
  - "Soft underbelly" of every DBMS
    - Sometimes not truly "optimal"

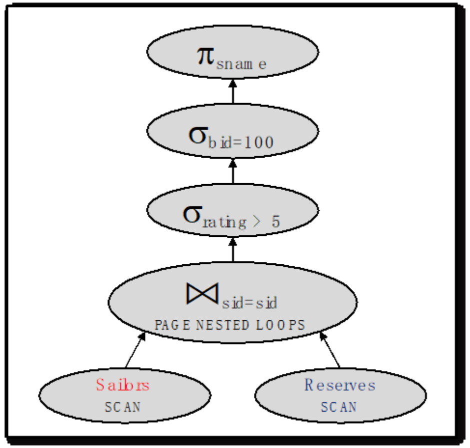


Query Optimization Overview

- Query block can be converted to relational algebra
- Rel. Algebra converts to 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
```

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



Query Optimization: The Components

- Three beautifully orthogonal concerns:
  - Plan space:
    - for a given query, what plans are considered?
  - Cost estimation:
    - how is the cost of a plan estimated?
  - Search strategy:
    - how do we “search” in the “plan space”?

Query Optimization: The Goal

- Optimization goal:
  - Ideally: Find the plan with least actual cost.
  - Reality: Find the plan with least estimated cost.
    - And try to avoid really bad actual plans!

Today

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

Relational Algebra Equivalences: Selections

- 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)

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:
  - $\pi_{a1}(R) \equiv \pi_{a1}(\ldots(\pi_{a1, \ldots, an-1}(R))\ldots)$ (cascade)

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:
  - $\pi_{a1}(R) \equiv \pi_{a1}(\ldots(\pi_{a1, \ldots, an-1}(R))\ldots)$ (cascade)
- Cartesian Product
  - $R \times (S \times T) \equiv (R \times S) \times T$ (associative)
  - $R \times S \equiv S \times R$ (commutative)

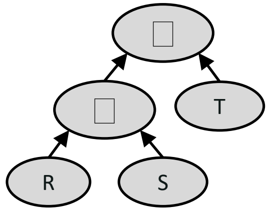
Are Joins Associative and Commutative?

- 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)
  - $(S \bowtie_{b=b} T) \bowtie_{a=a} R \not\equiv S \bowtie_{b=b} (T \bowtie_{a=a} R)$ (*not* legal!!)
    - $(S \bowtie_{b=b} T) \bowtie_{a=a} R \not\equiv S \bowtie_{b=b} (T \times R)$ (*not* the same!!)
  - $(S \bowtie_{b=b} T) \bowtie_{a=a} R \; \alpha \; S \bowtie_{b=b \wedge a=a} (T \times R)$ (the same!!)

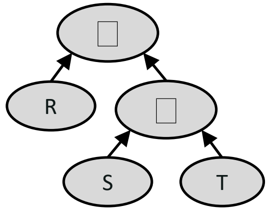
Join ordering, again

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

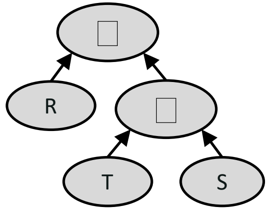
```
SELECT *
  FROM R, S, T
 WHERE R.a = S.a
   AND S.b = T.b;
```



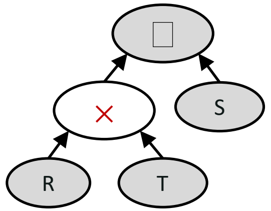
$(R \bowtie_{a=a} S) \bowtie_{b=b} T$



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



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



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

Some Common Heuristics: Selections

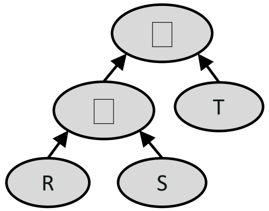
- Selection cascade and pushdown
  - Apply selections as soon as you have the relevant columns
  - Ex:
    - $\pi_{sname}\ (\sigma_{(bid=100\ \wedge\ rating\ >\ 5)}\ (Reserves \bowtie_{sid=sid} Sailors))$
    - $\pi_{sname}\ (\sigma_{bid=100}\ (Reserves) \bowtie_{sid=sid} \sigma_{rating\ >\ 5}\ (Sailors))$

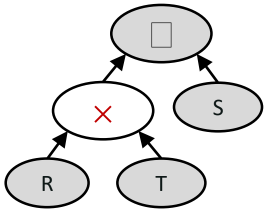
Some Common Heuristics: Projections

- Projection cascade and pushdown
  - Keep only the columns you need to evaluate downstream operators
  - Ex:
    - $\pi_{sname}\sigma_{(bid=100 \land rating > 5)}$ (Reserves $\bowtie_{sid=sid}$ Sailors)
    - π_{sname} (π_{sid}($\sigma_{bid=100}$ (Reserves)) $\bowtie_{sid=sid}$ $\pi_{sname,sid}$ ($\sigma_{rating > 5}$ (Sailors)))

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 $(R \bowtie S) \bowtie T$ over $(R \times T) \bowtie S$





Physical Equivalences

- Base table access,
  with single-table selections and projections
    - 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 Hash Join: even better than sort with 1 small table
        - Or Hybrid if you have it
- Non-Equijoins
    - Block Nested Loop

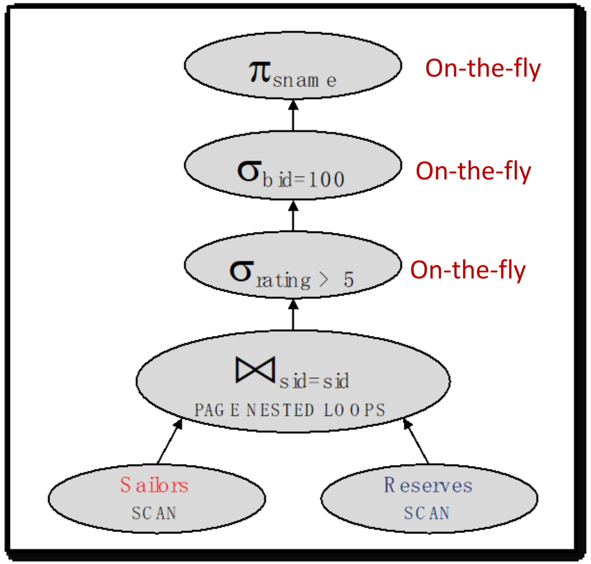
Schema for Examples

```
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
- Sailors:
  - Each tuple is 50 bytes long, 80 tuples per page, 500 pages.
  - Assume there are 10 different ratings
- Assume we have 5 pages to use for joins.

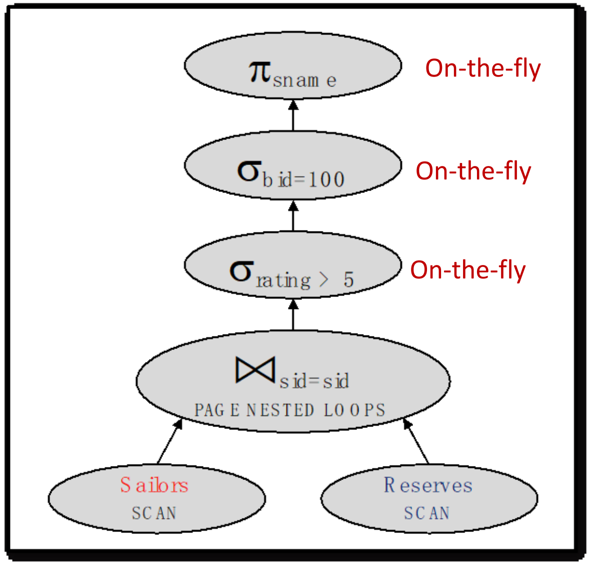
Motivating Example: Plan 1

- Here’ s a reasonable query plan:



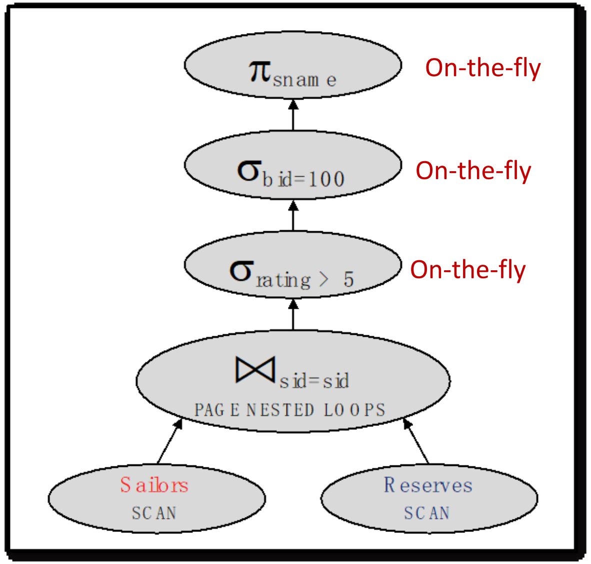
```
SELECT S.sname
  FROM Reserves R, Sailors S
 WHERE R.sid=S.sid
   AND R.bid=100
   AND S.rating>5
```

Motivating Example: Plan 1 Cost



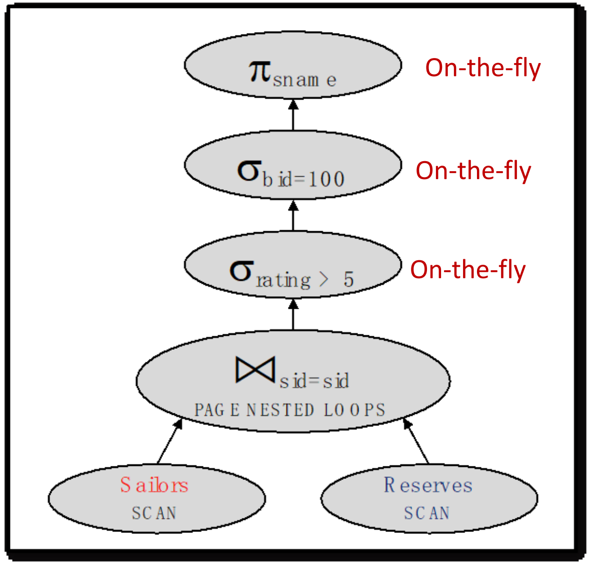
- Let’s estimate the cost:
- Scan Sailors (500 IOs)
- For each page of Sailors,
  Scan Reserves (1000 IOs)
- Total: 500 + 500*1000
  - 500,500 IOs

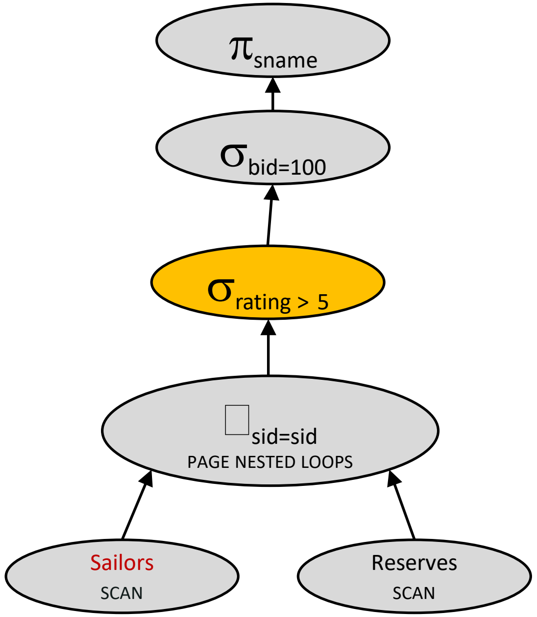
Motivating Example: Plan 1 Cost Analysis



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

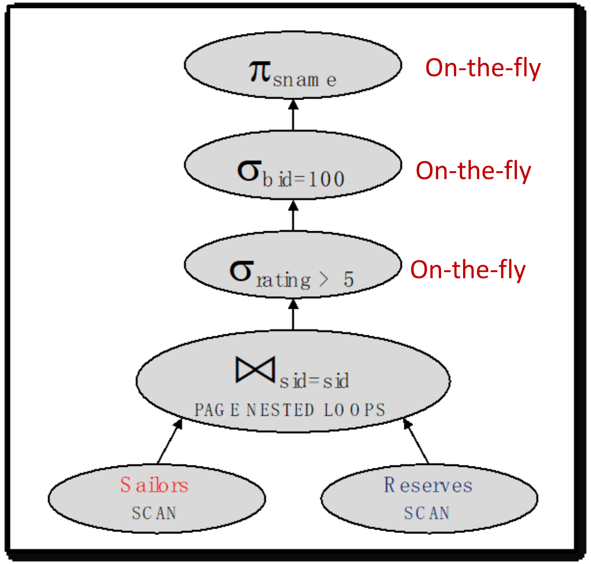
Selection Pushdown



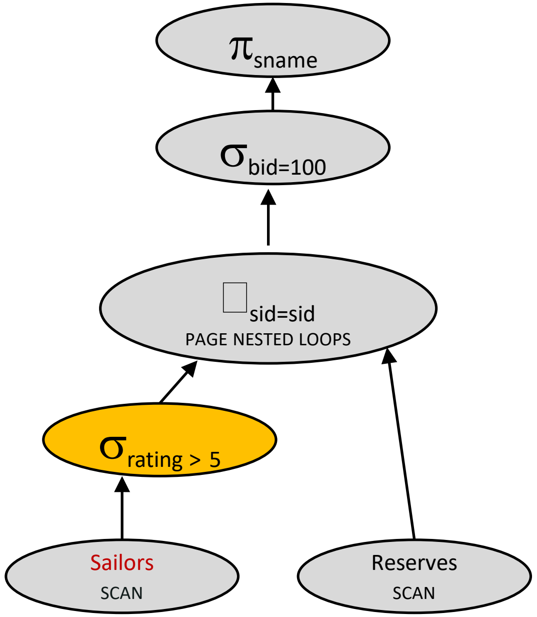


500,500 IOs

Selection Pushdown, cont



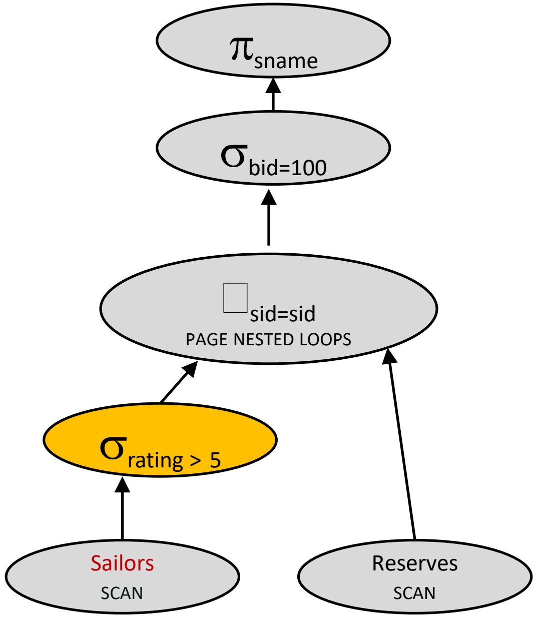
500,500 IOs



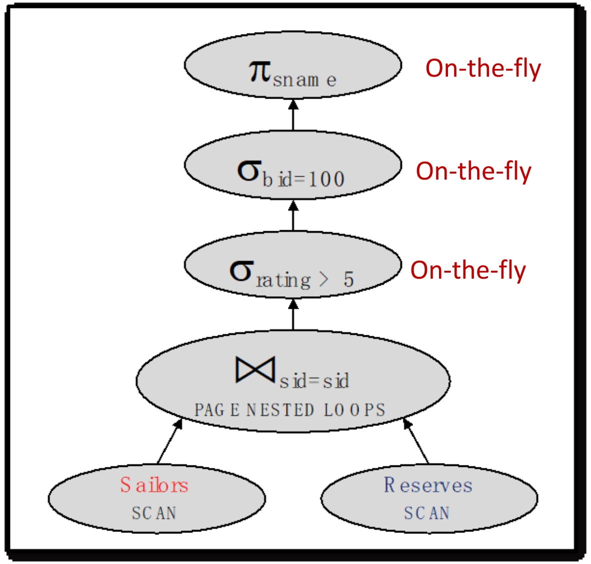
Cost?

Query Plan 2 Cost

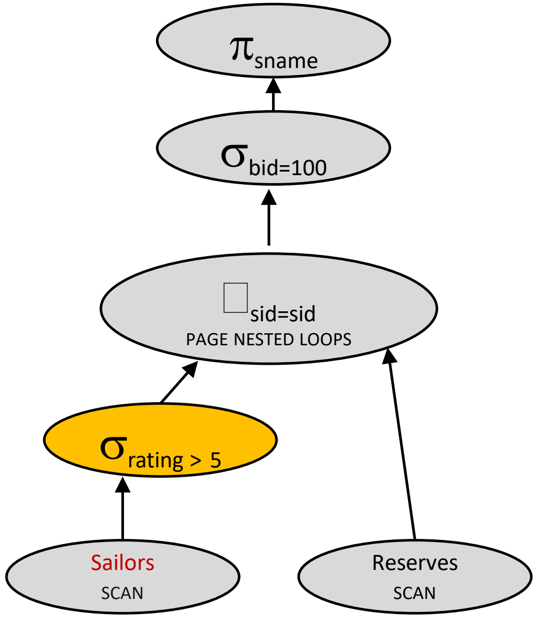
- Let’s estimate the cost:
- Scan Sailors (500 IOs)
- For each pageful of high-rated Sailors,
  Scan Reserves (1000 IOs)
- Total: 500 + ???*1000
- Total: 500 + 250*1000



Decision?

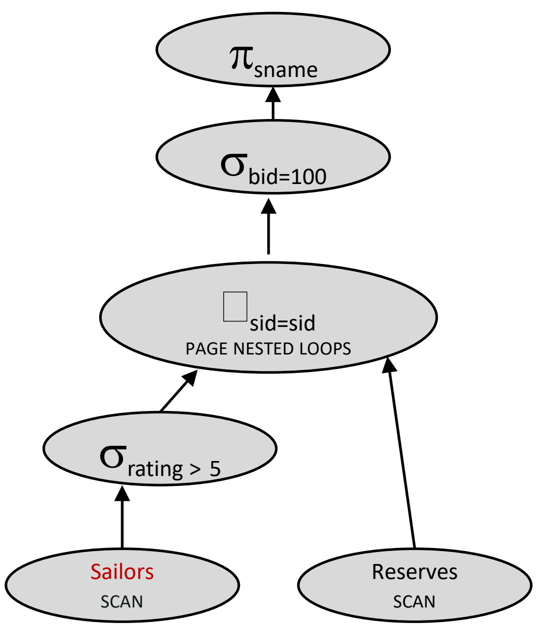


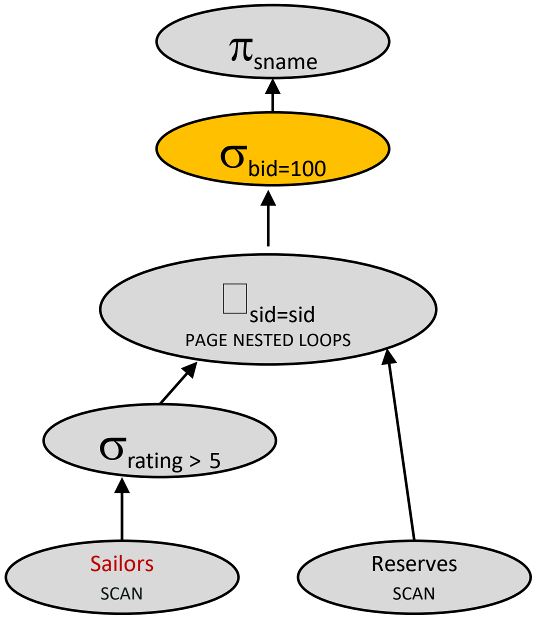
500,500 IOs



250,500 IOs

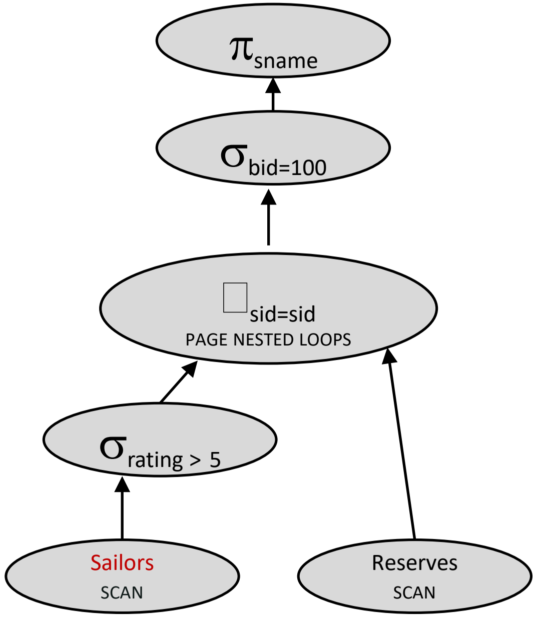
More Selection Pushdown



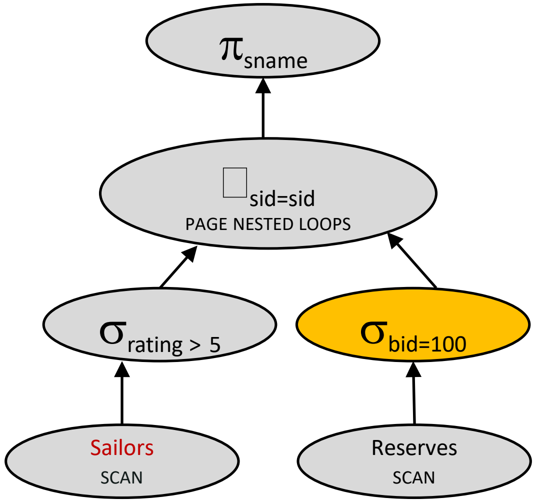


250,500 IOs

More Selection Pushdown, cont



250,500 IOs



Cost???

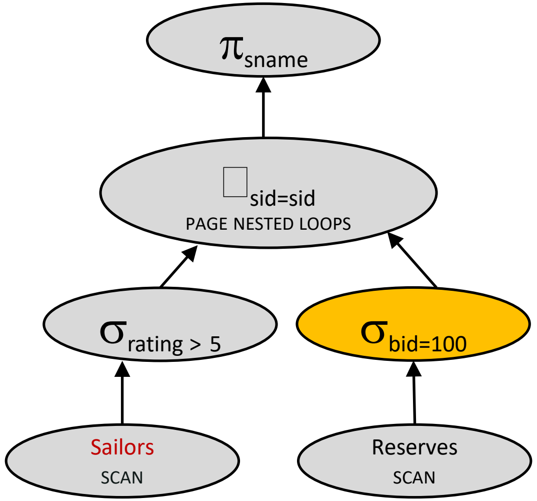
Query Plan 3 Cost Analysis

Let's estimate the cost:

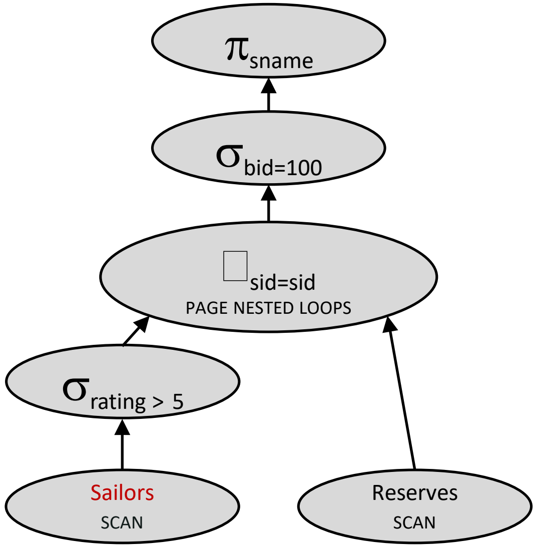
- Scan Sailors (500 IOs)
- For each pageful of high-rated Sailors,
  Do what? (??? IOs)

- Total: 500 + 250*???

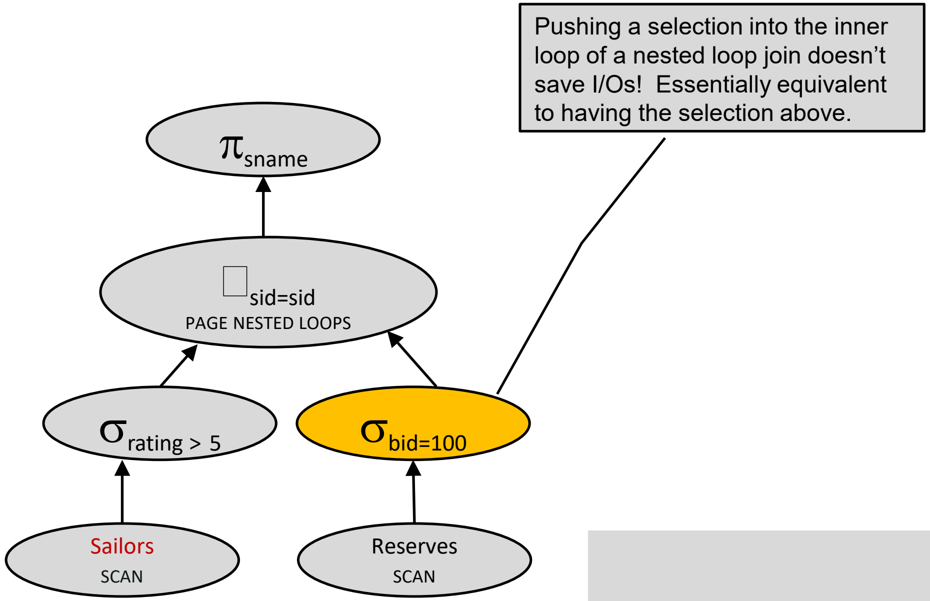
- Total: 500 + 250*1000!



More Selection Pushdown Analysis



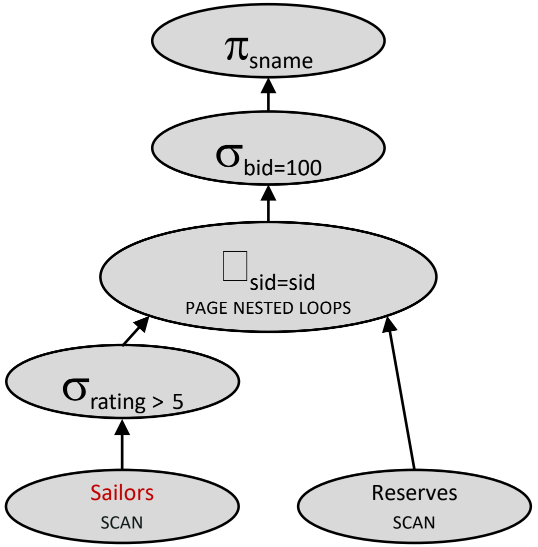
250,500 IOs



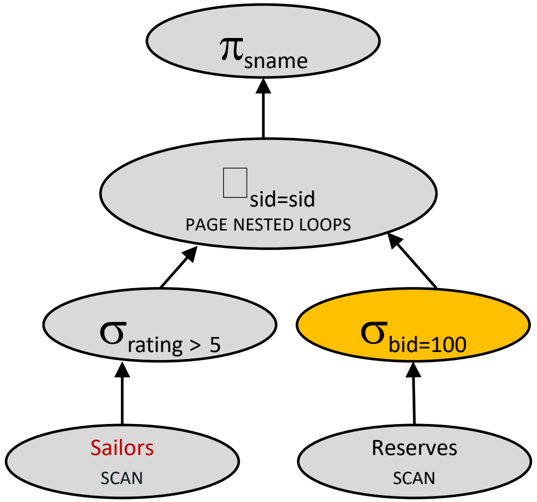
Pushing a selection into the inner loop of a nested loop join doesn't save I/Os! Essentially equivalent to having the selection above.

250,500 IOs

Decision 2

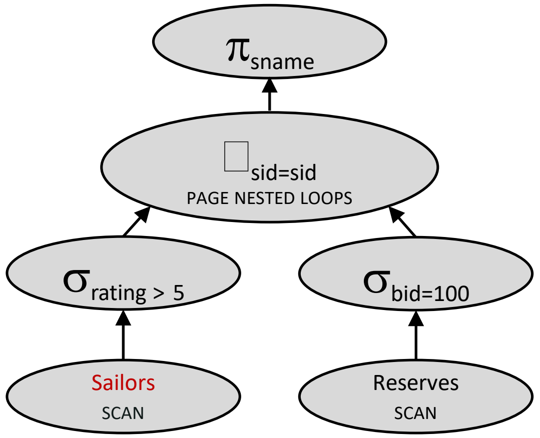


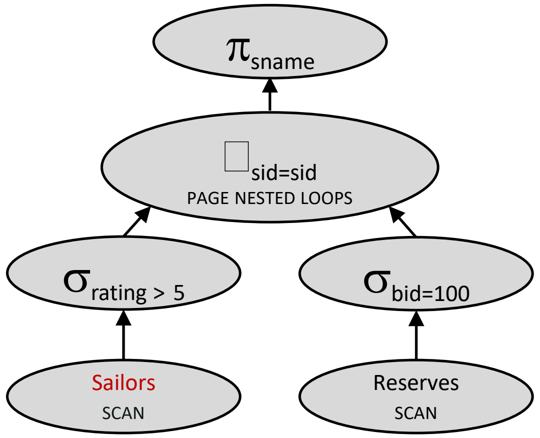
250,500 IOs



250,500 IOs

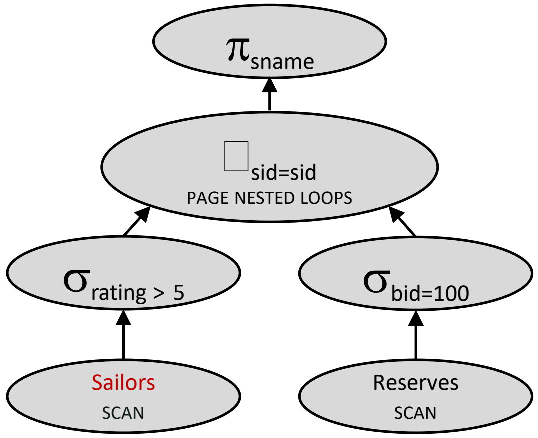
Join Ordering

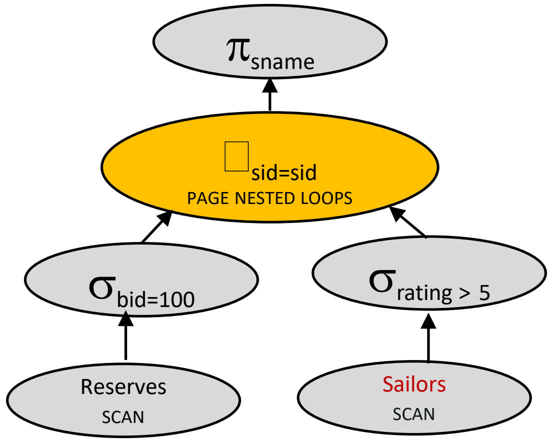




250,500 IOs

Join Ordering, cont

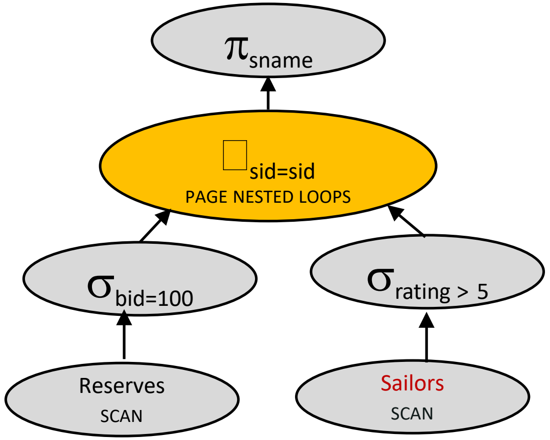




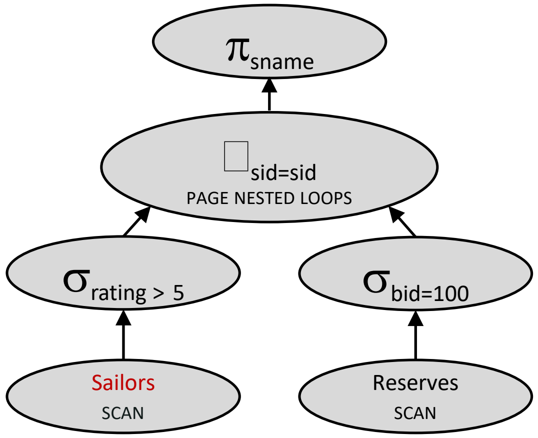
250,500 IOs

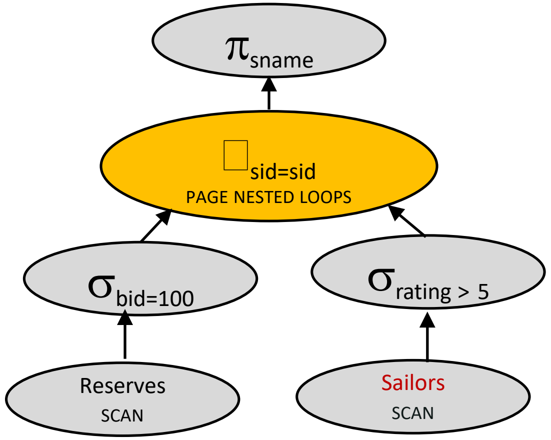
Query Plan 4 Cost

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



Decision 3

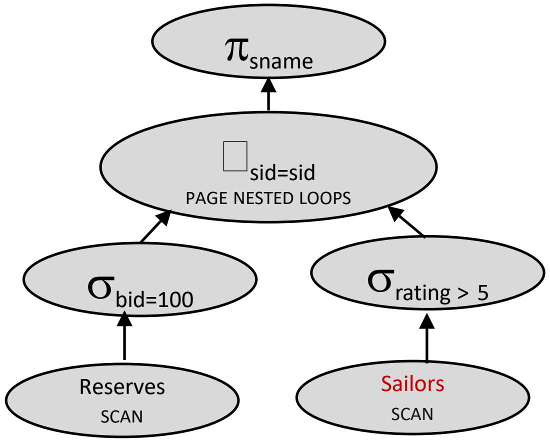




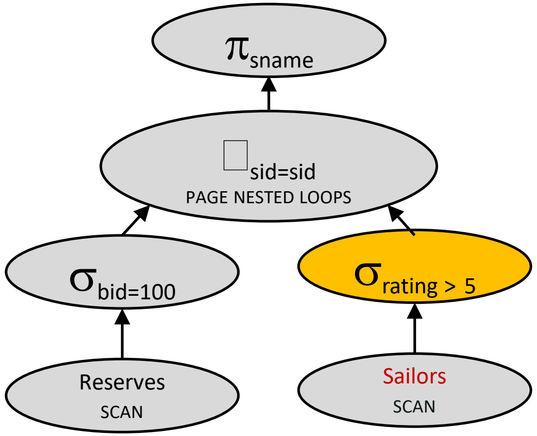
250,500 IOs

6000 IOs

Materializing Inner Loops

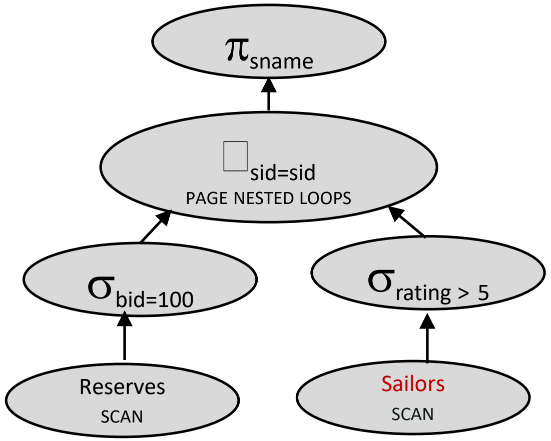


6000 IOs

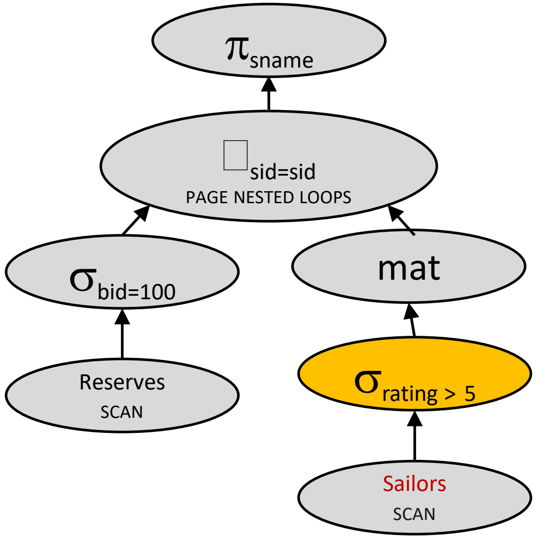


Cost???

Materializing Inner Loops, cont



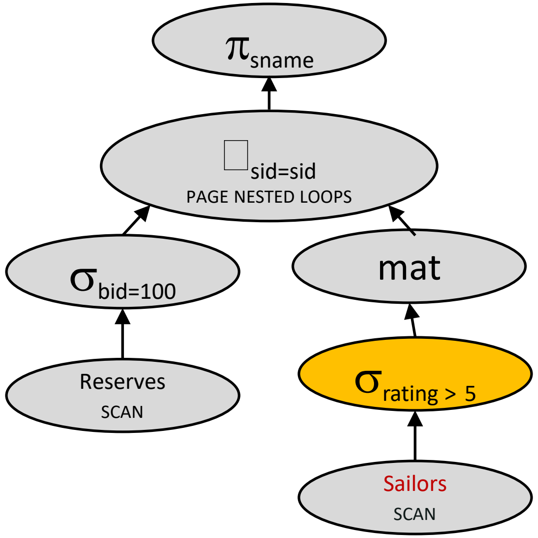
6000 IOs



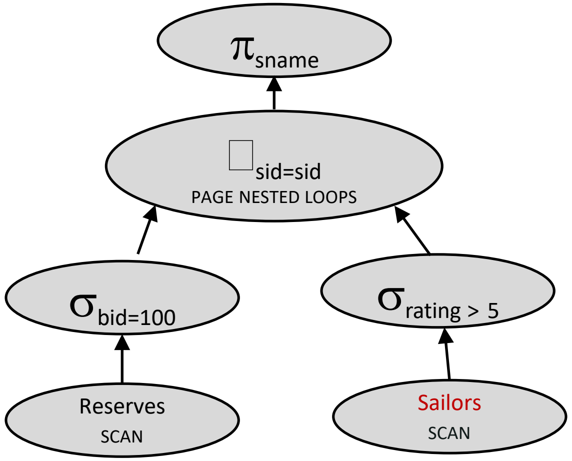
Cost???

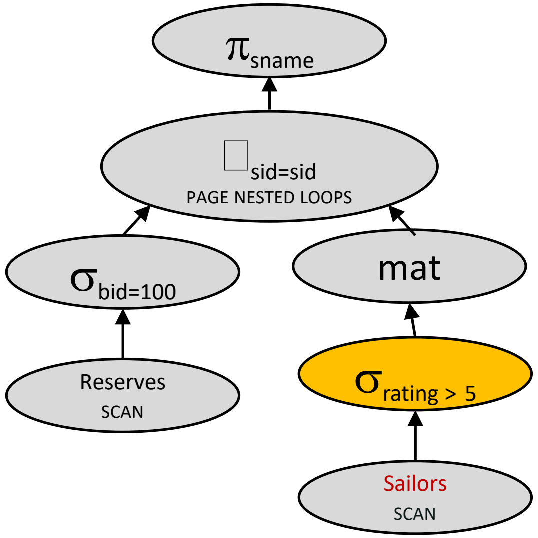
Plan 5 Cost Analysis

- 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)



Materializing Inner Loops, cont.

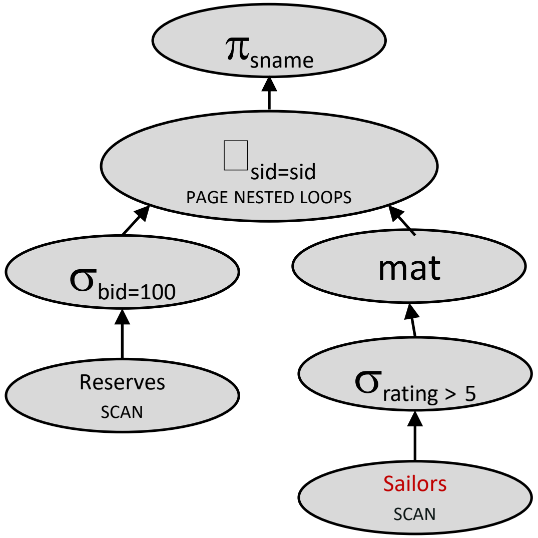




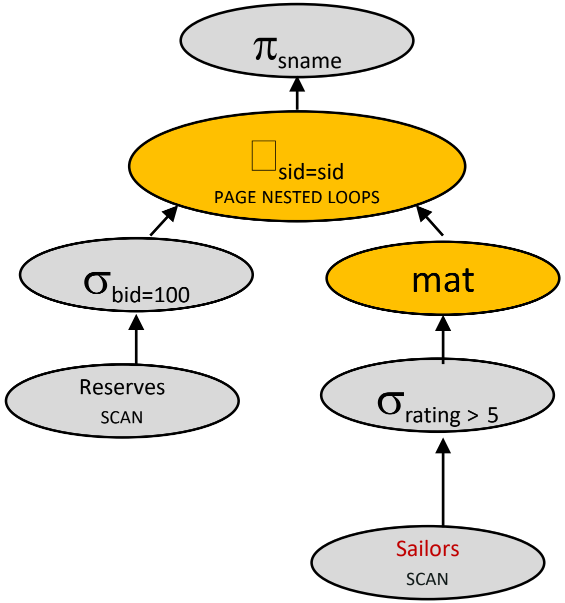
6000 IOs

4250 IOs

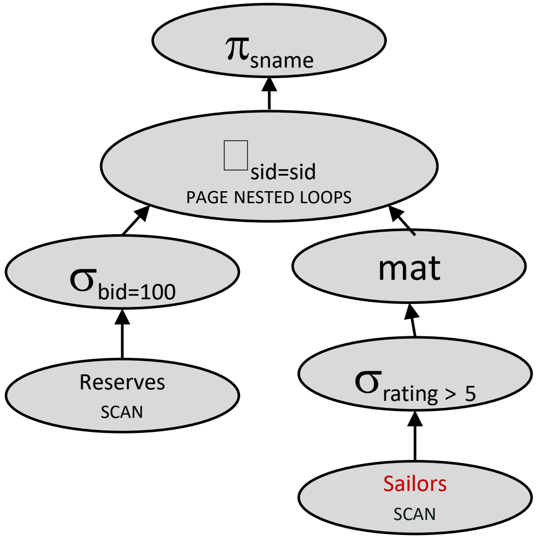
Join Ordering Again



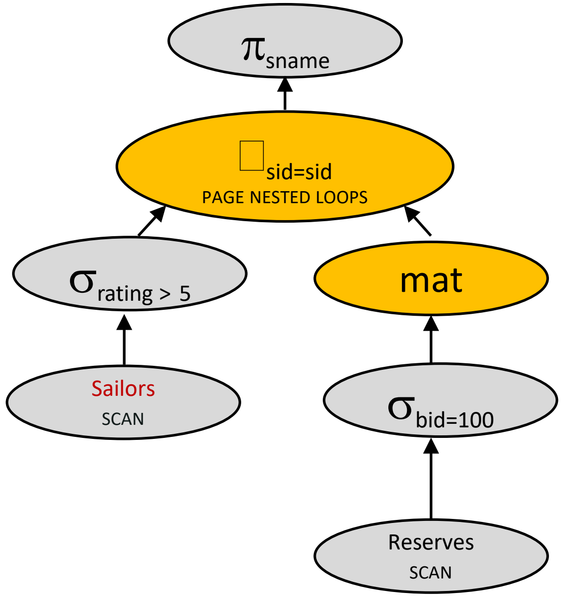
4250 IOs



Join Ordering Again, Cont



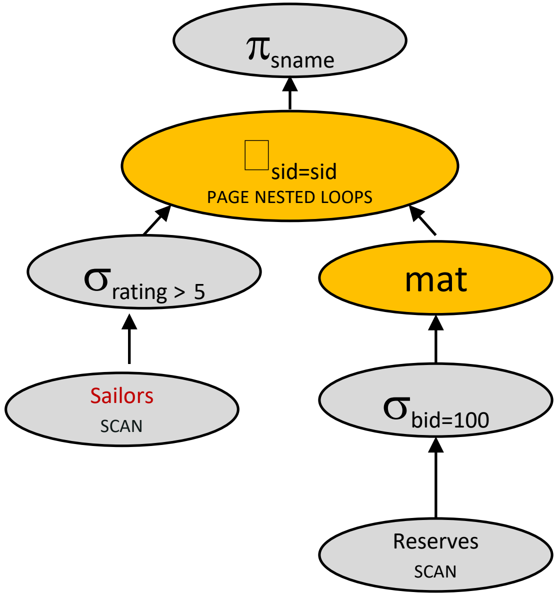
4250 IOs



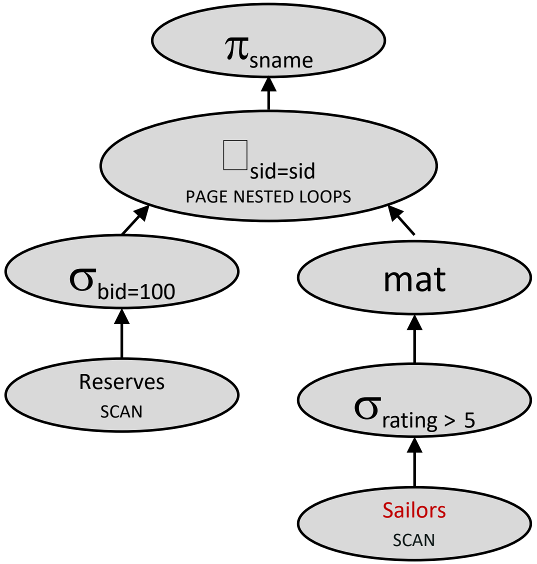
Cost???

Plan 6 Cost Analysis

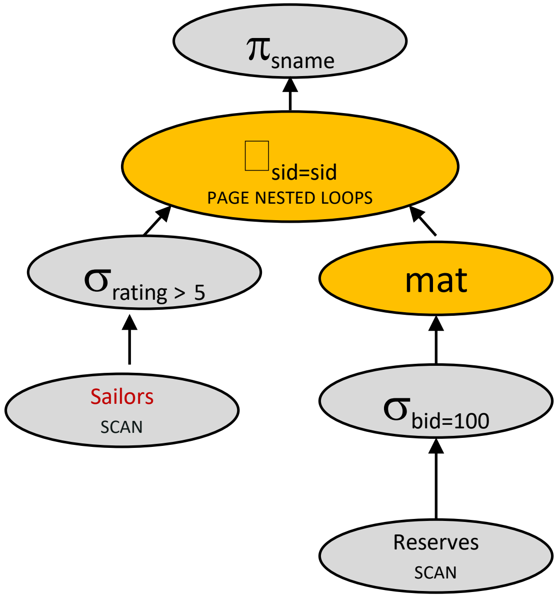
- Let's estimate the cost:
- Scan Sailors (500 IOs)
- Scan Reserves (1000 IOs)
- Materialize Temp table T1 (??? IOs)
- For each pageful of high-rated Sailors,
  Scan T1 (??? IOs)
- Total: 500 + 1000 + ??? + 250*???
- 500 + 1000 +10 +(250 *10)



Decision 4

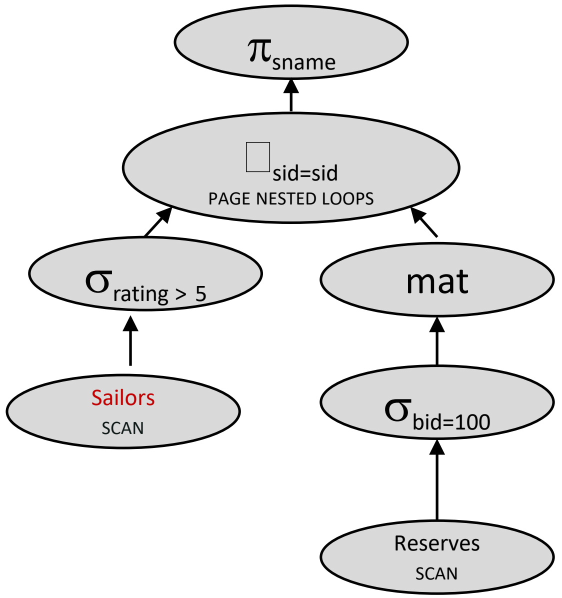


4250 IOs

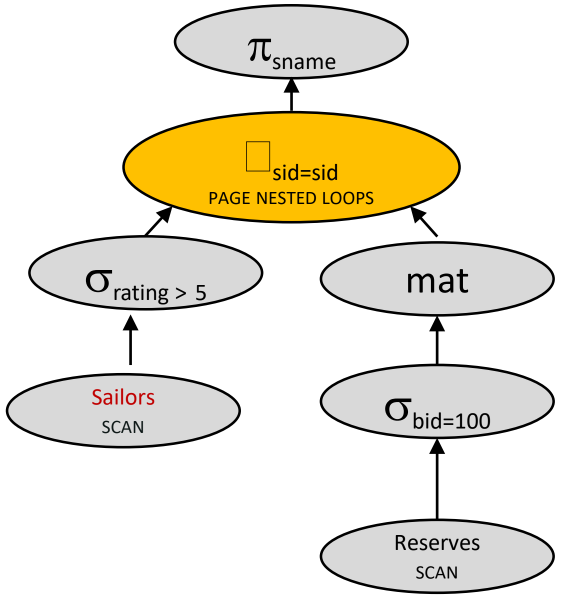


4010 IOs

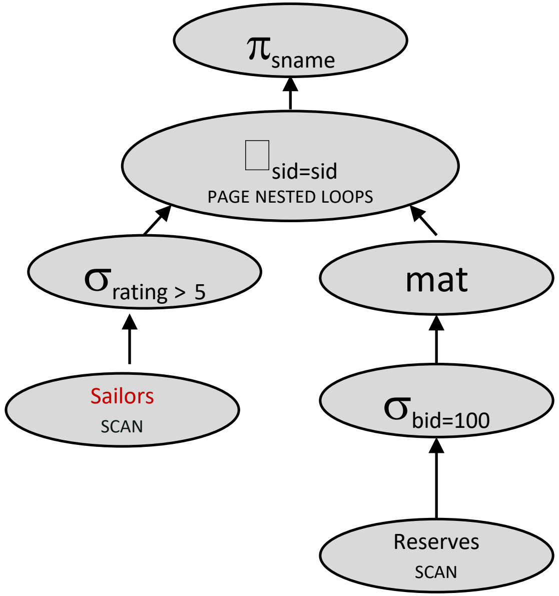
Join Algorithm



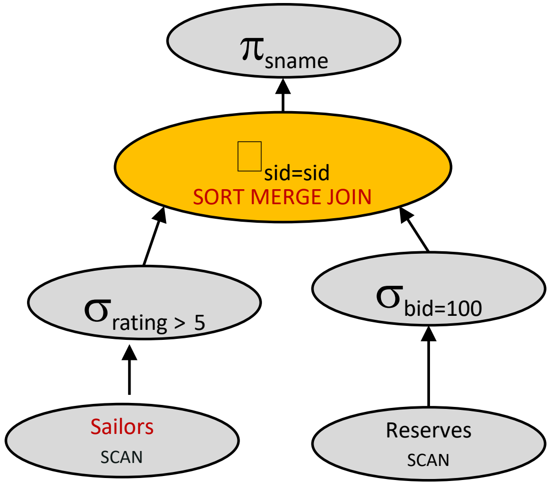
4010 IOs



Join Algorithm, cont.



4010 IOs



Cost???

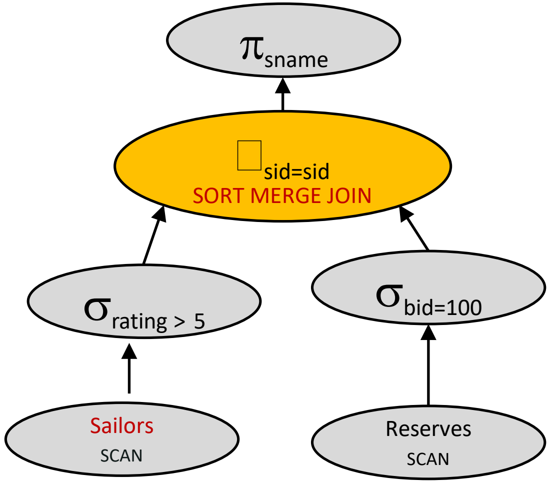
Query Plan 7 Cost Analysis

- 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 with $\log_4$?

- Merge (10+250) = 260
- Total:



Query Plan 7 Cost Analysis Part 2

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

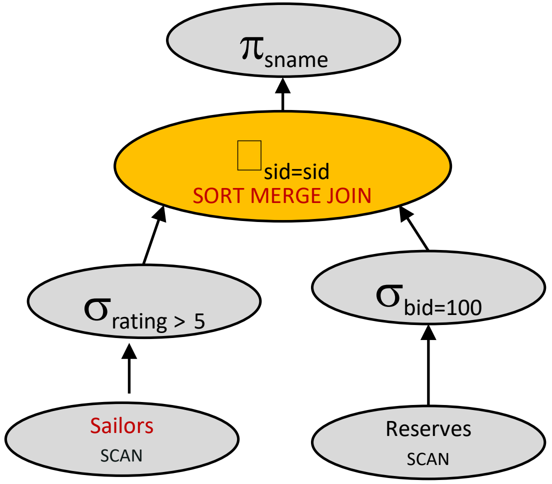
- Sort

  - 2 passes for reserves
    pass 0 = 10 to write, pass 1 = 2*10 to read/write

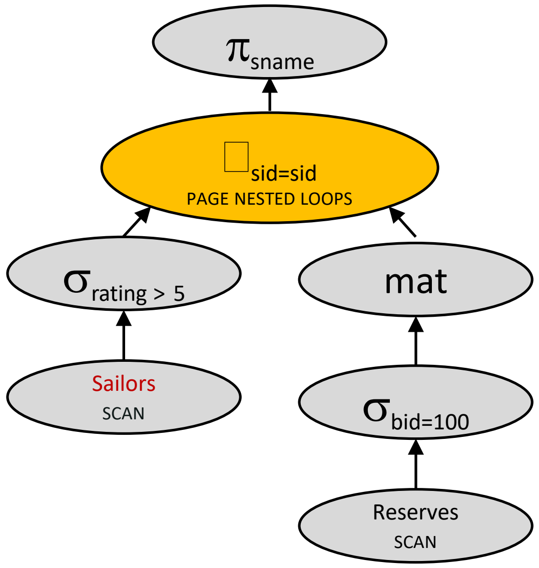
  - 4 passes for sailors
    pass 0 = 250 to write, pass 1,2,3 = 2*250 to read/write

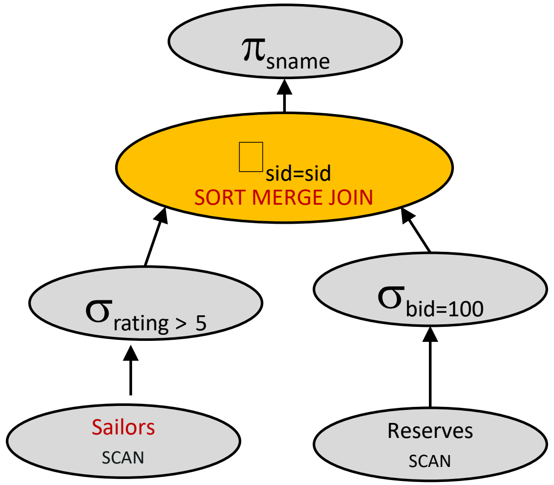
- Merge (10+250) = 260

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



Decision 5

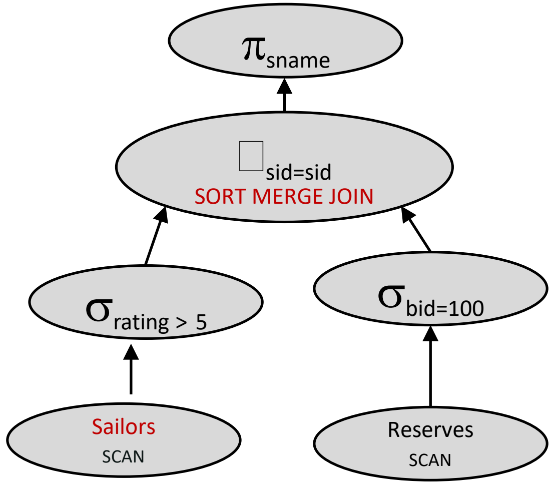


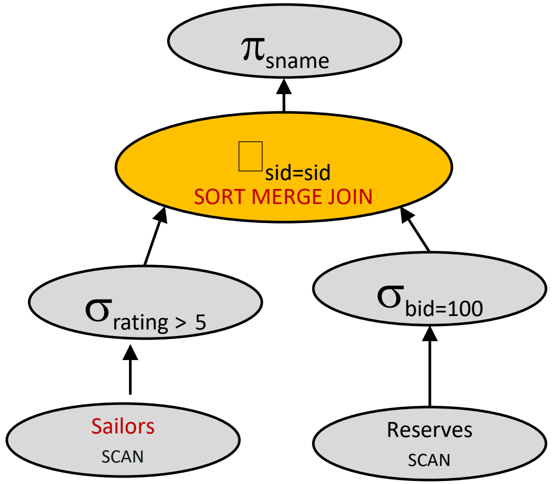


4010 IOs

3540 IOs

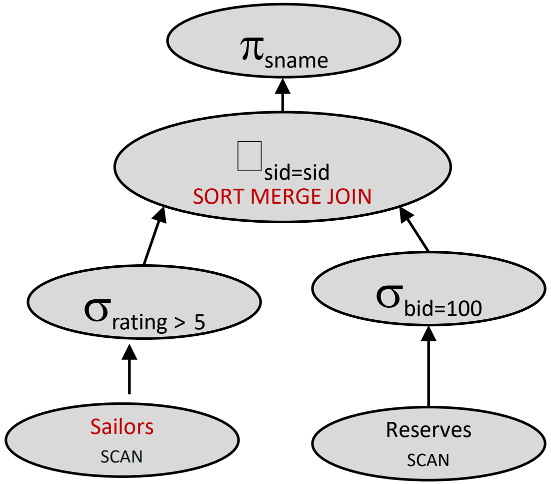
Textbook considers this:



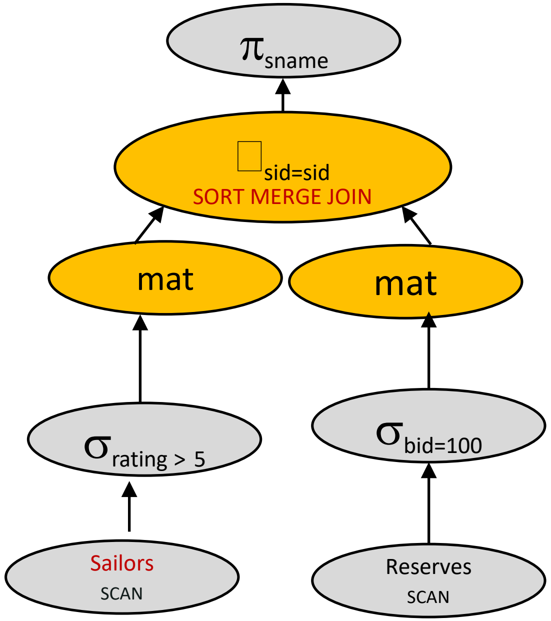


3540 IOs

Textbook considers this, cont:



3540 IOs



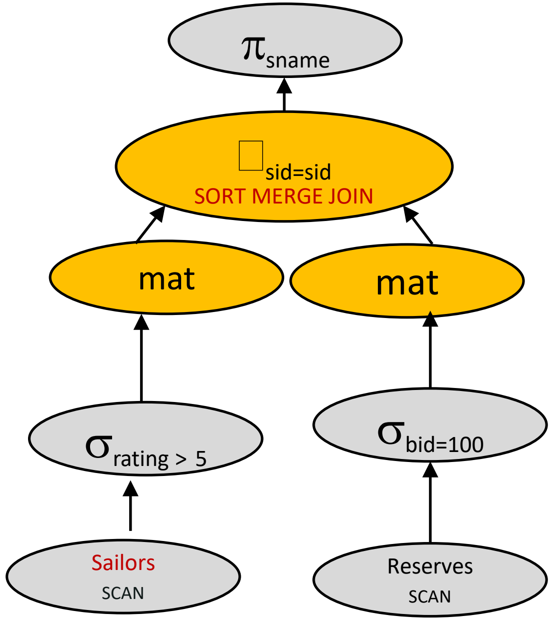
Cost???

Plan 8 Cost Analysis

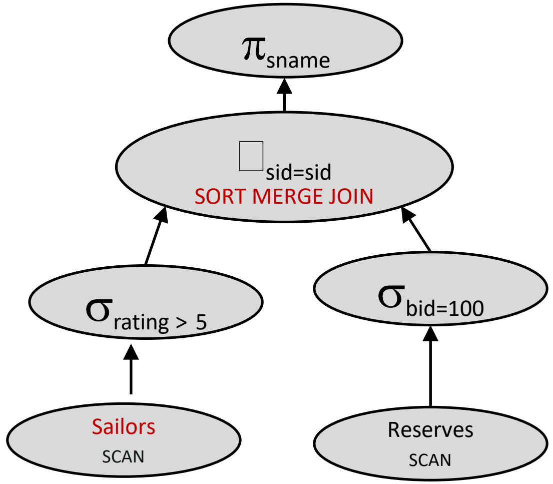
- With 5 buffers, cost of plan:
- Scan Sailors (500), write T1 (250)
- Scan Reserves (1000), write T2 (10)
- Sort T1 (???)
- Sort T2 (???)

- How many passes for each sort?
  - 2 passes for reserves (2*2*10 to read/write)
  - 4 passes for sailors (4*2*250 to read/write)

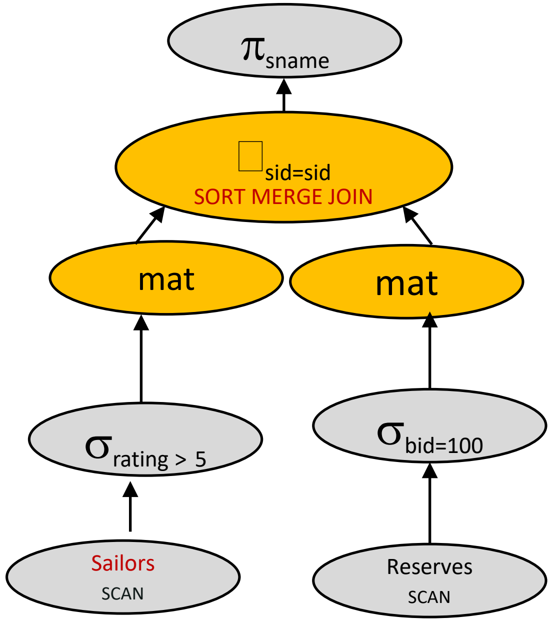
- Merge (10+250) = 260
- Total:
  1000 + 10 + 500 + 250 + 2*2*10 +
  4*2*250 + merge (10+250) = 4060



Decision 6

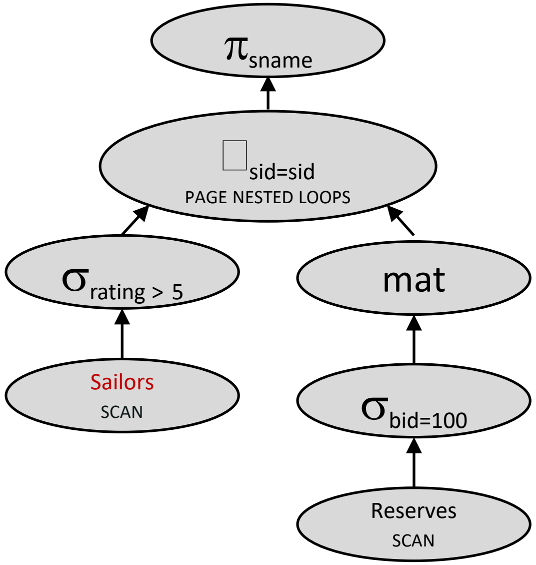


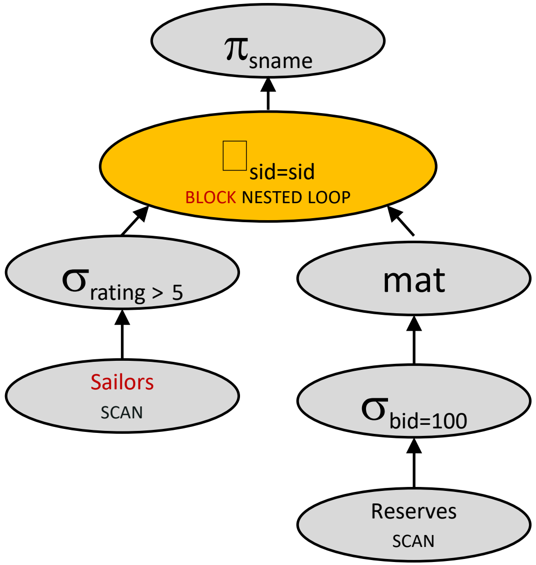
3540 IOs



4060 IOs

Join Algorithm Again, Again





4010 IOs

Cost???

Query 9 Cost Analysis

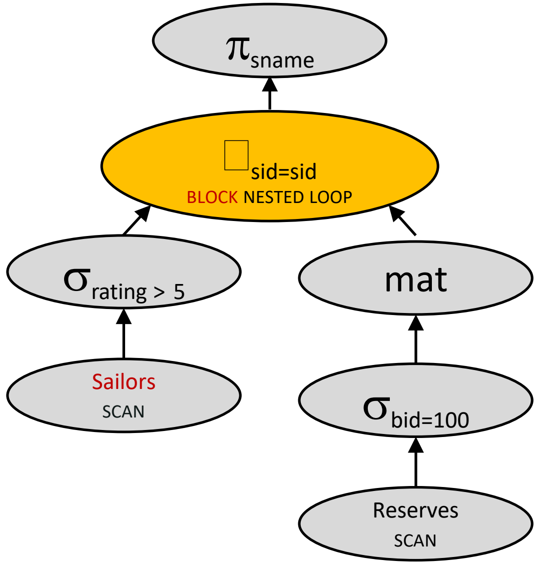
- With 5 buffers, cost of plan:

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

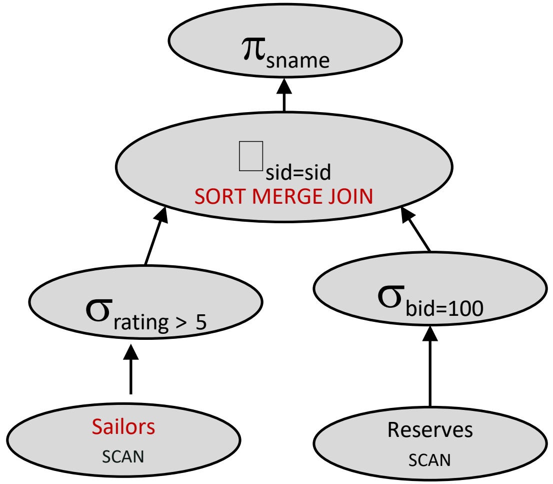
- Write Temp T1 (10)
- For each blockful of high-rated sailors
-   Loop on T1 (??? * 10)

- Total:

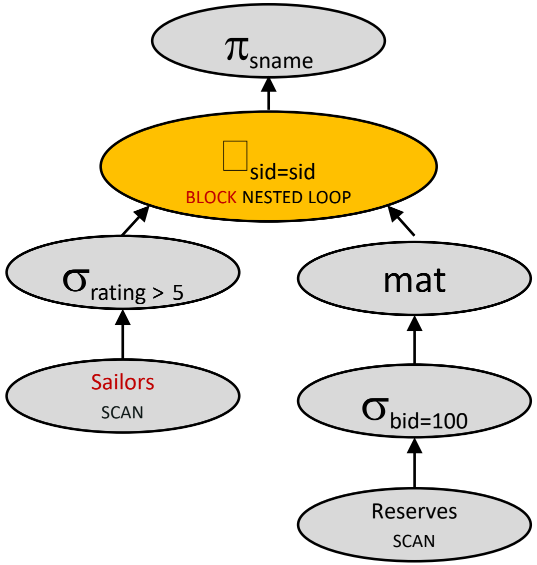
500 + 1000 +10 +(ceil(250/3) *10) = 500 + 1000 +10 +(84 *10) =



Decision 7

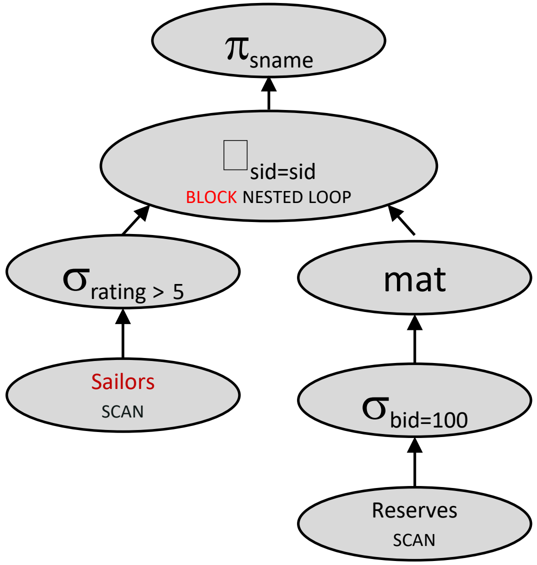


3540 IOs



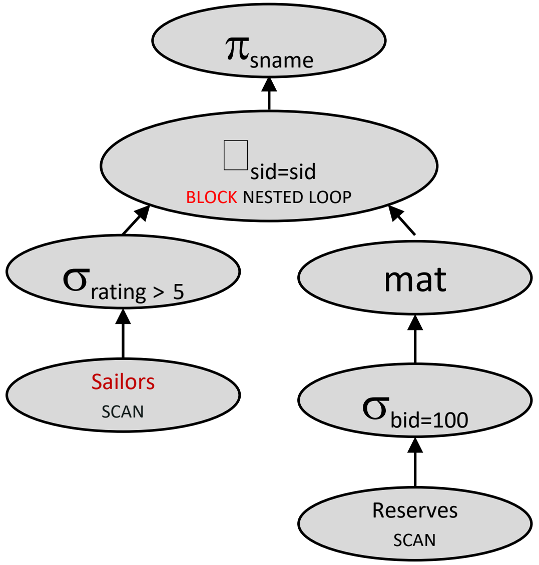
2350 IOs

Projection Cascade & Pushdown

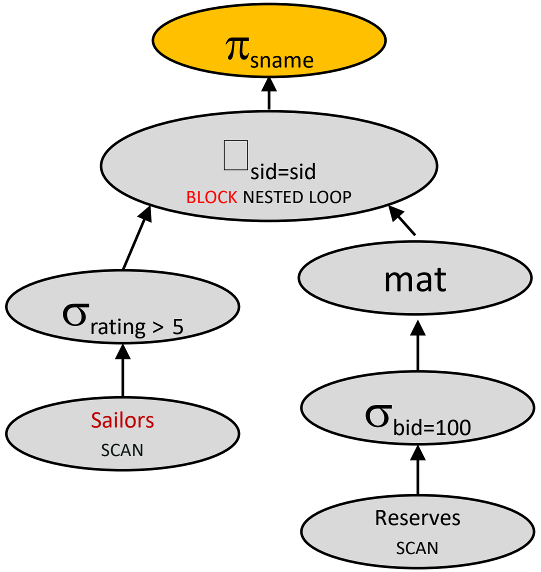


2350 IOs

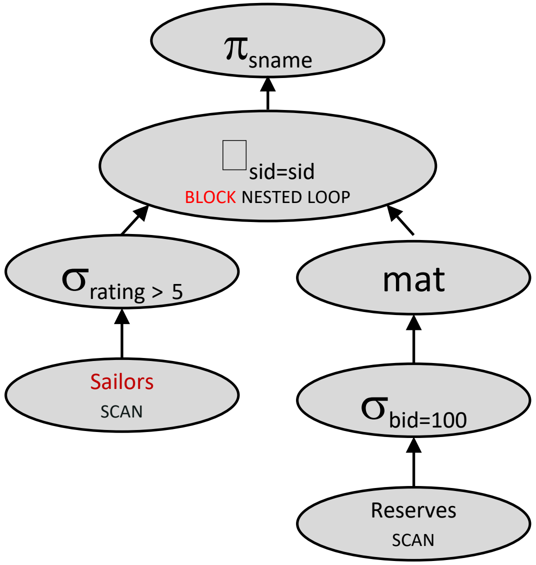
Projection Cascade & Pushdown, cont



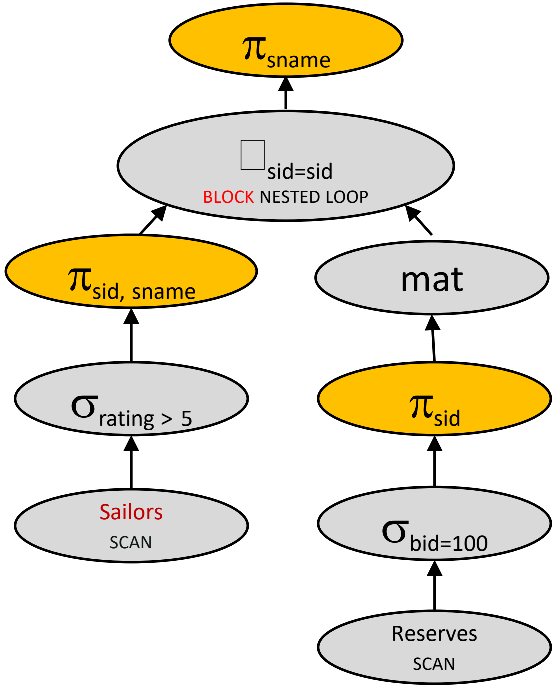
2350 IOs



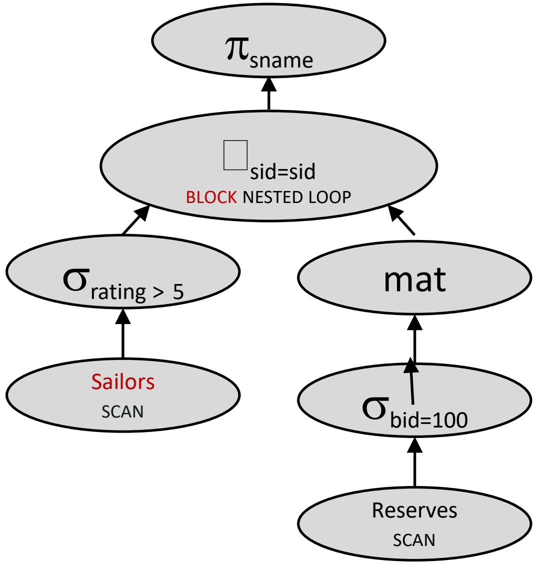
Projection Cascade & Pushdown, cont



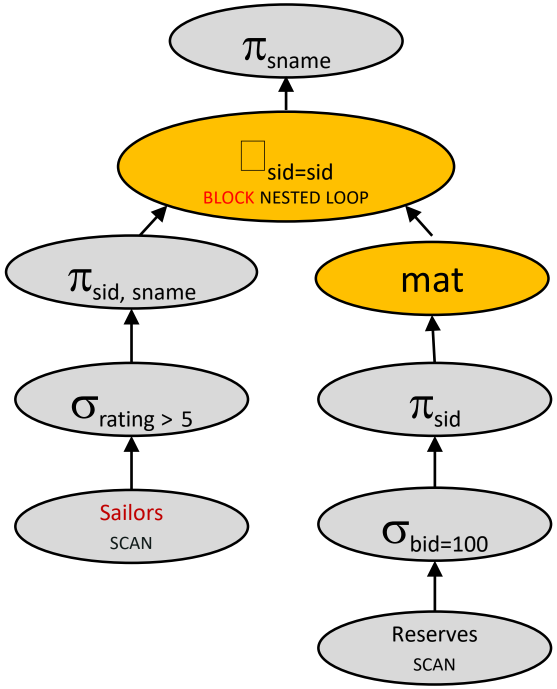
2350 IOs



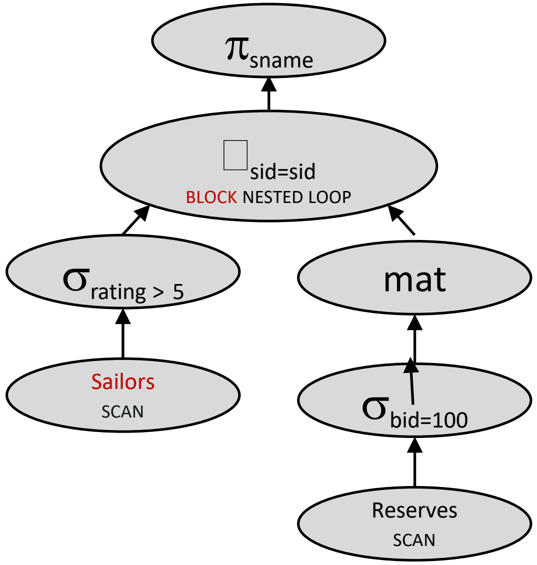
With Join Reordering, no Mat

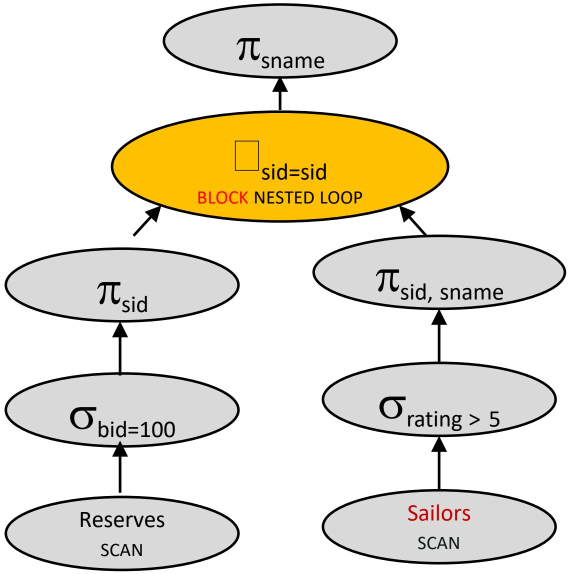


2350 IOs



With Join Reordering, no Mat cont

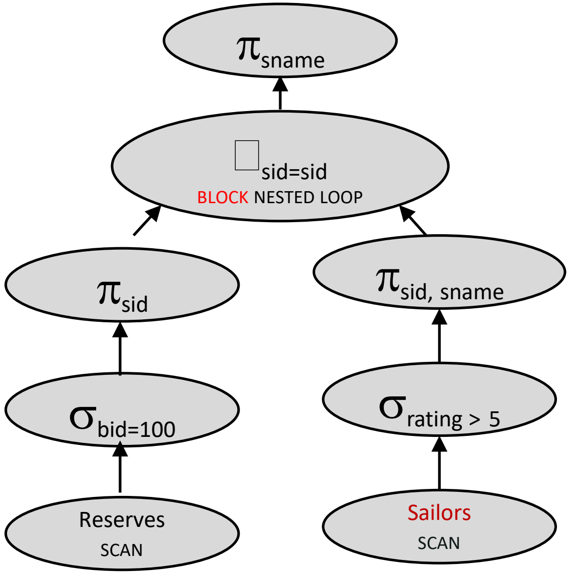




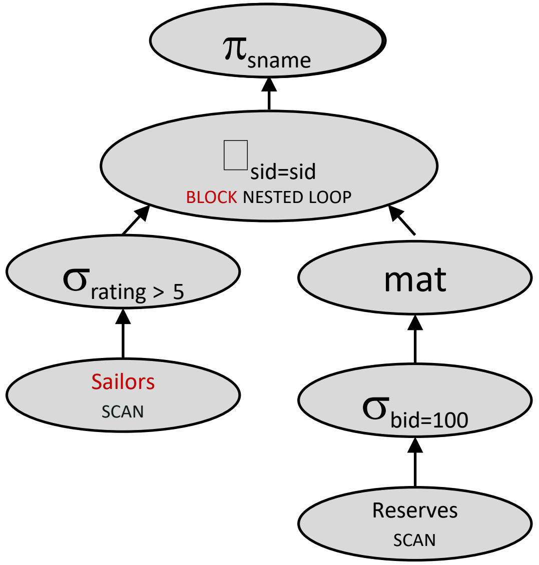
2350 IOs

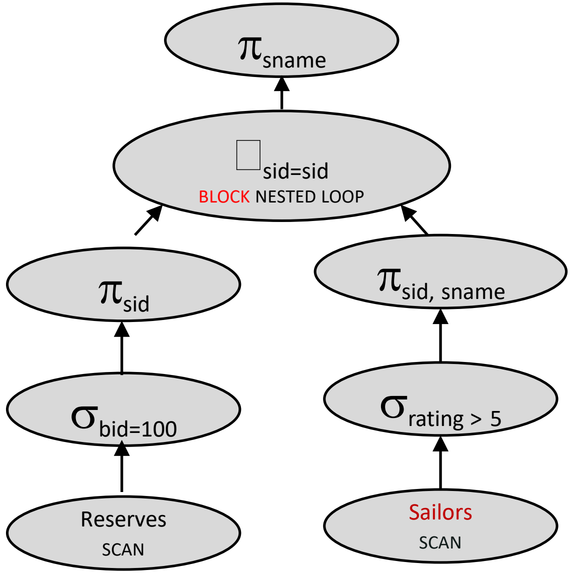
Plan 11 Cost Analysis

- 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)
- Loop on Sailors (??? * 500)
- Total: 1500



With Join Reordering, no Mat, cont.



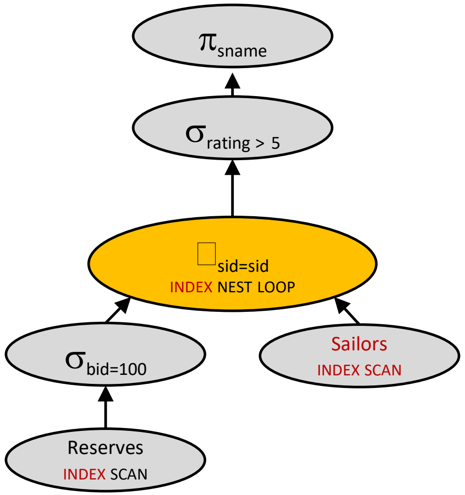


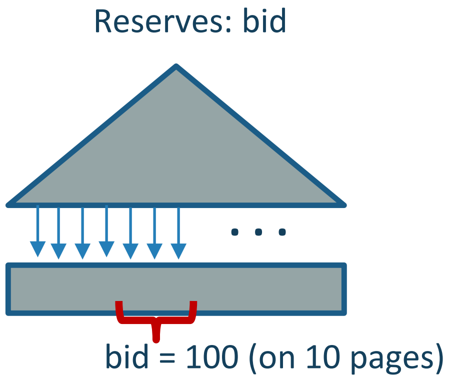
2350 IOs

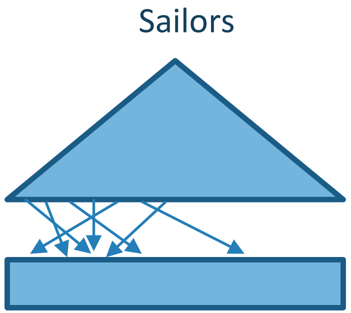
1500 IOs

How About Indexes?

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

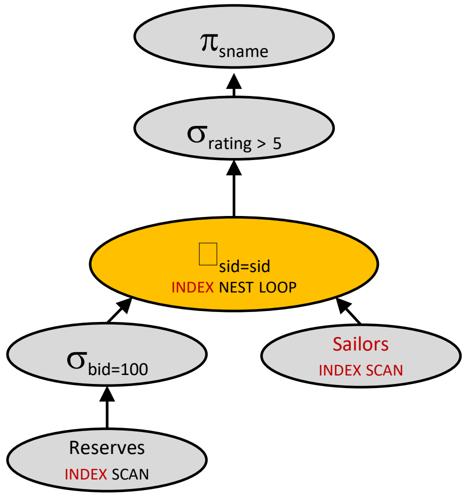






Index Cost Analysis

- **No projection pushdown to left** for π_{sname}
  - Projecting out unnecessary fields from outer of Index NL doesn't make an I/O difference.
- **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, unclustered index on sid OK


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

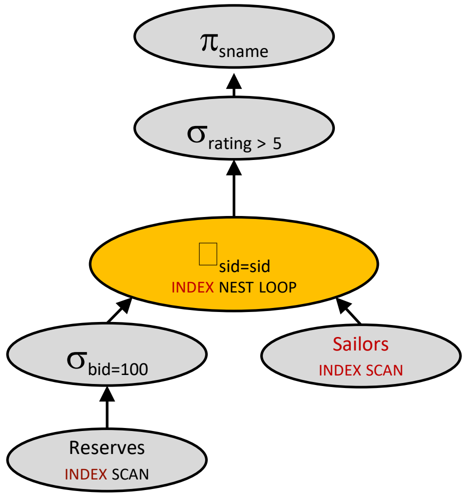
Index Cost Analysis Part 2

- 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
  get matching Sailors tuple (1 IO)
  (recall: 100 Reserves per page, 1000 pages)

- 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

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
    - Hence AI
  - We will see soon how optimizers make simplifying assumptions