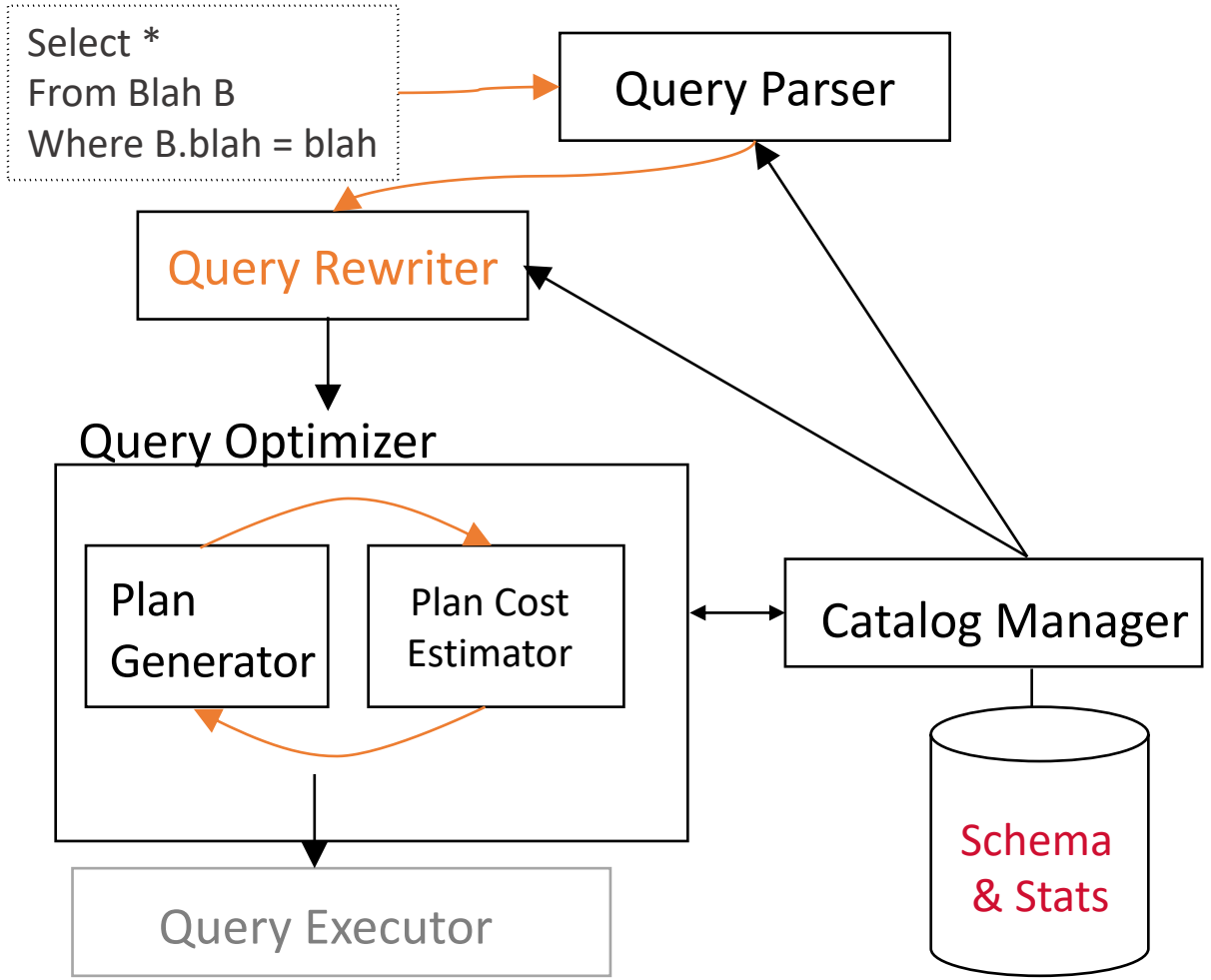
Introduction to Database Systems

2023-Fall

4. Database Management Systems （3/5）

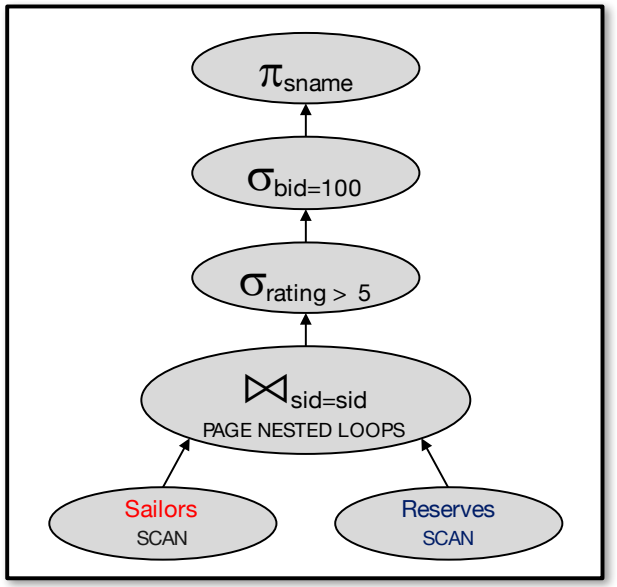
Query Parsing & Optimization



- ***Query parser***
  - Checks correctness, authorization
  - Generates a parse tree
  - Straightforward
- ***Query rewriter***
  - Converts queries to canonical form
    - flatten views
    - subqueries into fewer query blocks
  - Weak spot in many open-source DBMSs
- ***"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

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
```

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

$(Reserves \bowtie Sailors)$

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!

Relational Algebra Equivalences

- ***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?

- 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!!)
  - $(S \bowtie_{b=b} T) \bowtie_{a=a} R \equiv S \bowtie_{b=b \wedge a=a} (T \times R)$ (the same!!)

Some Common Heuristics: Selections

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

Some Common Heuristics: Projections

- Projection cascade and pushdown
  - Keep only the columns you need to evaluate downstream operators
  - Ex:
    - $\pi_{sname}\sigma_{(bid=100 \wedge rating > 5)}$ (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 relation small and the other indexed properly
  - Sort-Merge Join: good with small memory, equal-size tables
- Non-Equijoins
  - Block Nested Loop

Schema for Examples

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

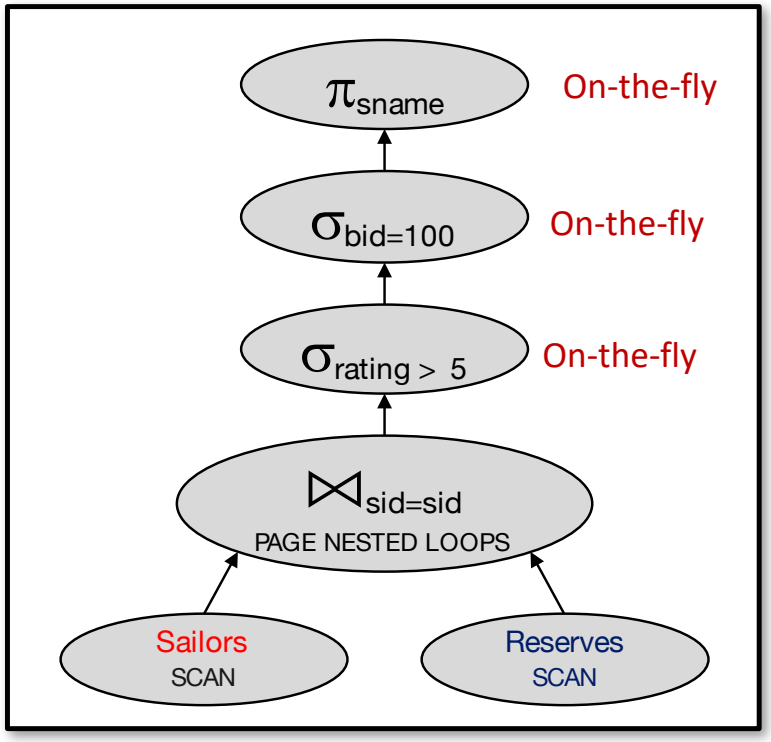
Reserves (*sid*: integer, *bid*: integer, *day*: date, *rname*: text)

- 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.

Example 1

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

- Here's a reasonable query plan:



- 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

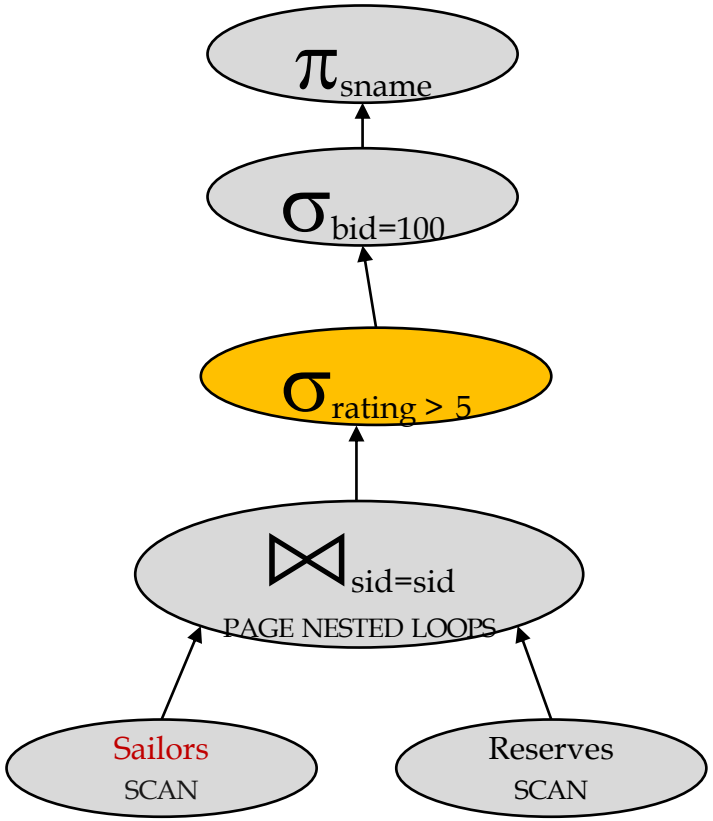
Misses several opportunities: ***selections could be 'pushed' down****, and* ***no use made of indexes***
Goal of optimization: ***Find faster plans that compute the same answer.***

4.3 Query Optimization

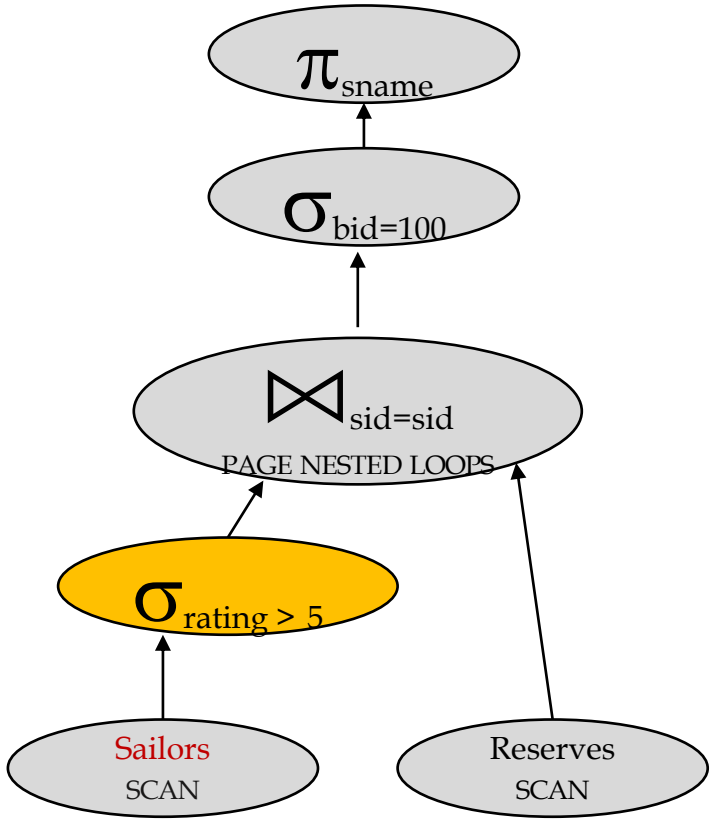
“Rewrite” the query statements submitted by user first, and then decide the most effective operating method and steps to get the result. The goal is to gain the result of user’s query with the lowest cost and in shortest time.

- **Algebra Optimization**
- **Operation Optimization**

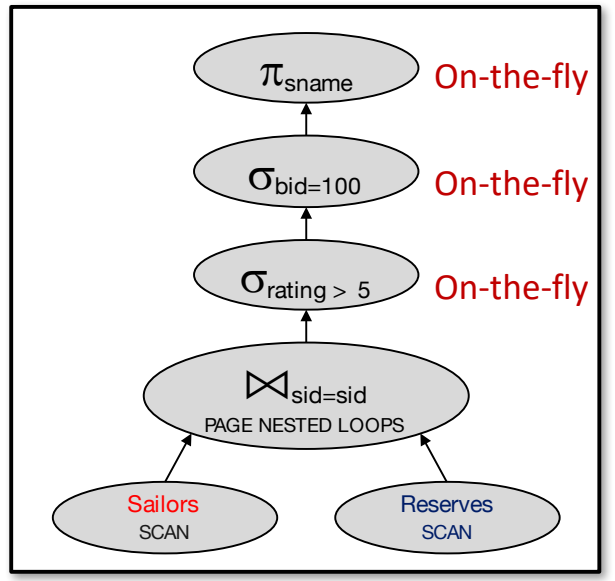
Plan1: Selection Pushdown



500,500 IOs



Cost?
250,500 IOs

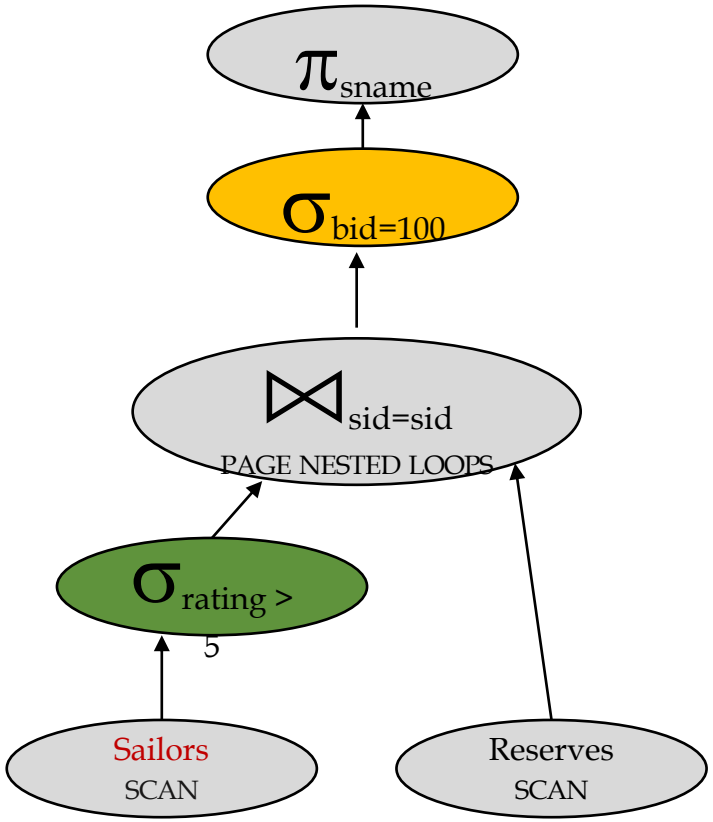


- Let's estimate the cost:
- Scan Sailors (500 IOs)
- For each page of high-rated Sailors, Scan Reserves (1000 IOs)

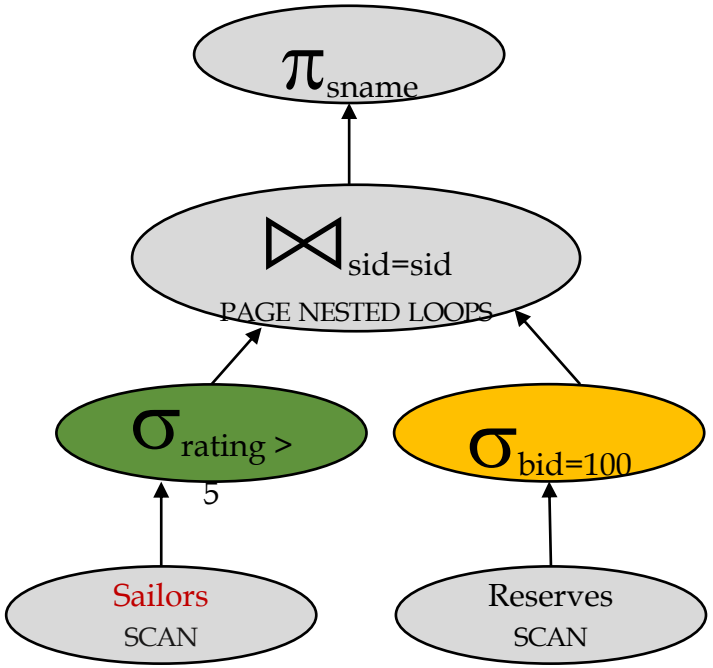
Total: 500 + ???*1000

Total: 500 + 250*1000

Plan2: More Selection Pushdown



250,500 IOs

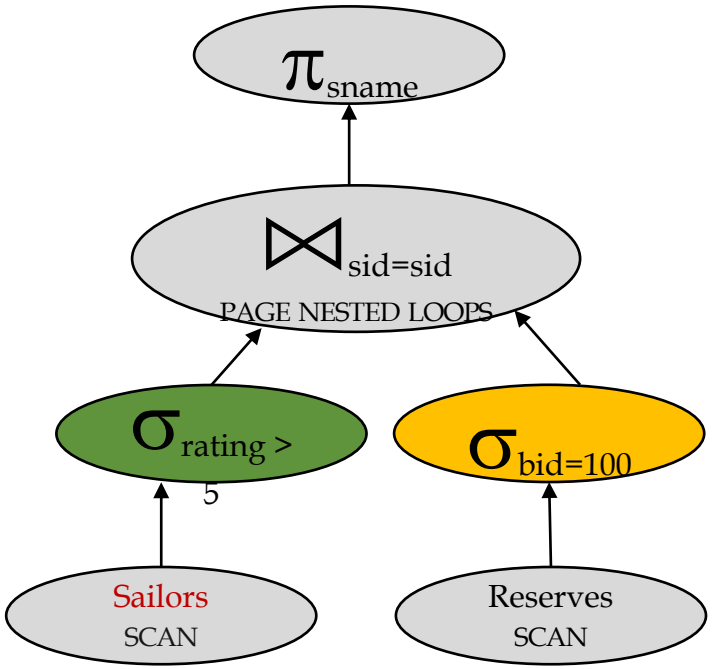


Cost???

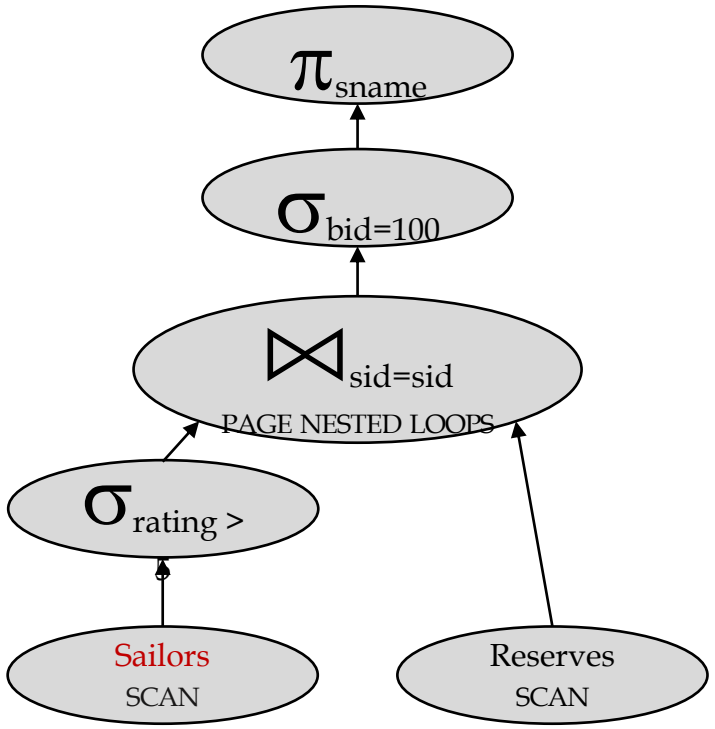
Plan 2 Cost Analysis

Let's estimate the cost:

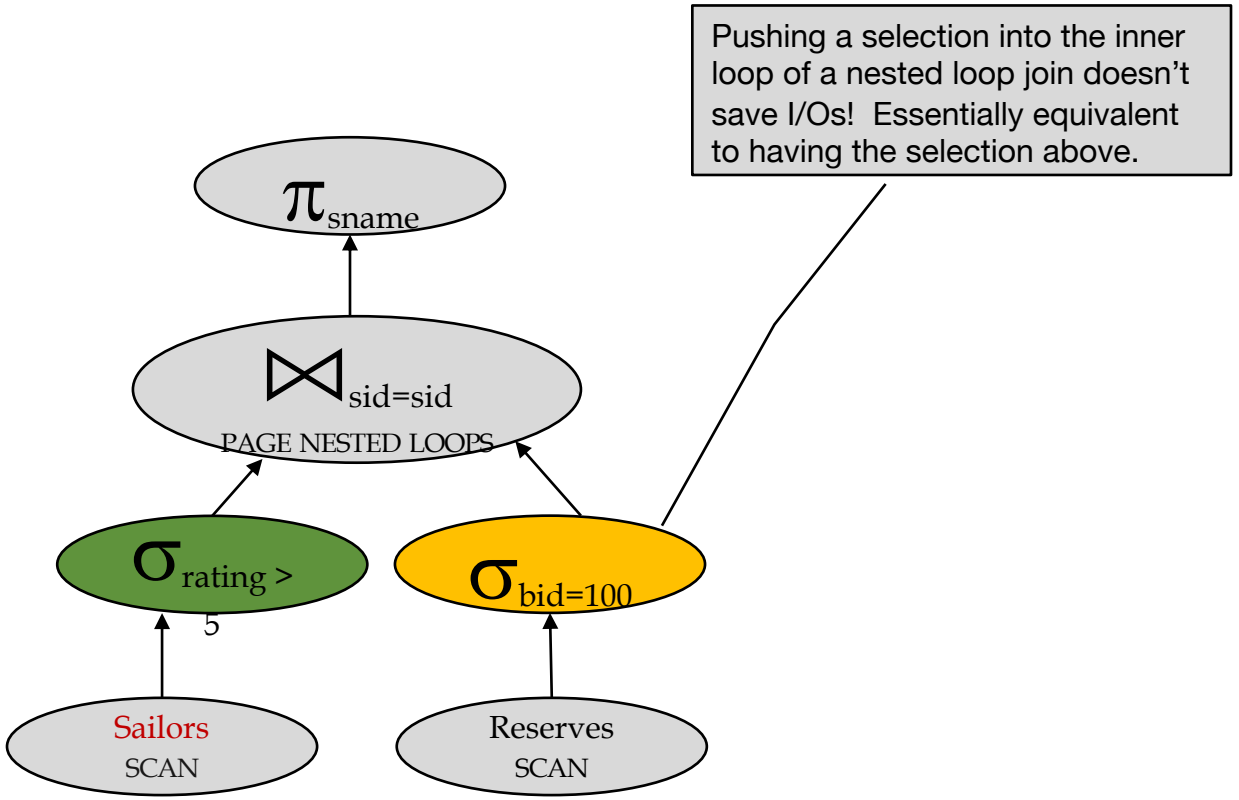
- Scan Sailors (500 IOs)
- For each pageful of high-rated Sailors,
  Do what? (??? IOs)
- Total: 500 + 250*???
- Total: 500 + 250*1000!



More Selection Pushdown Analysis

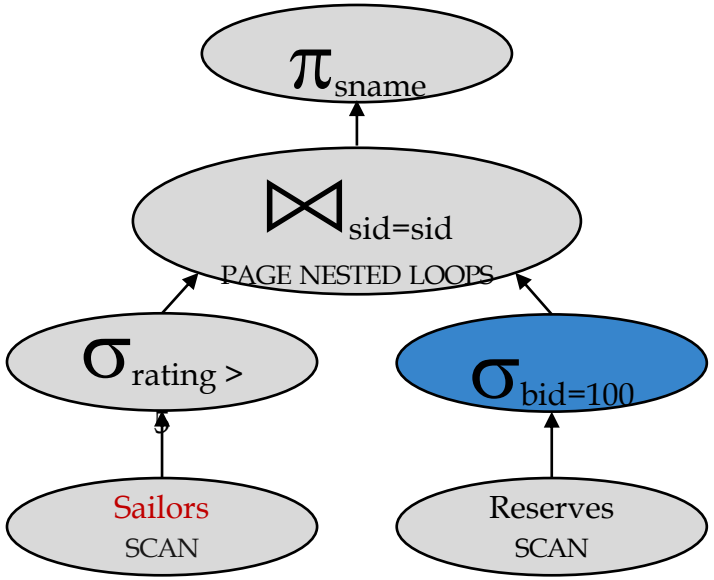


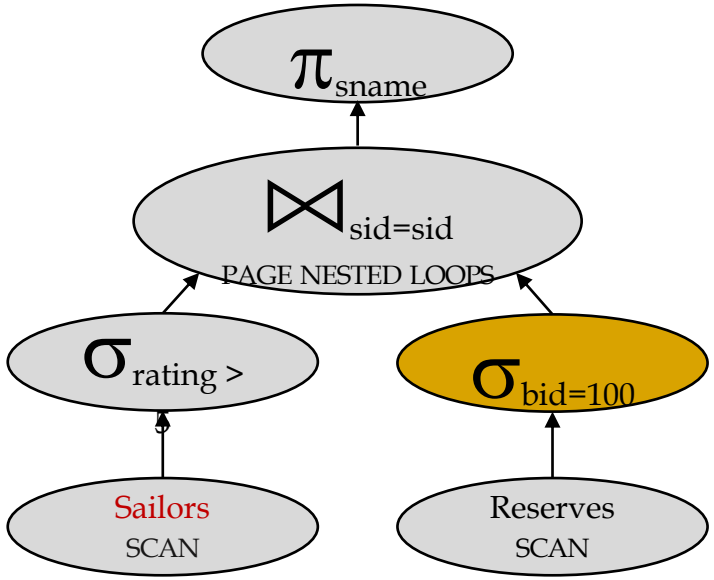
250,500 IOs



250,500 IOs

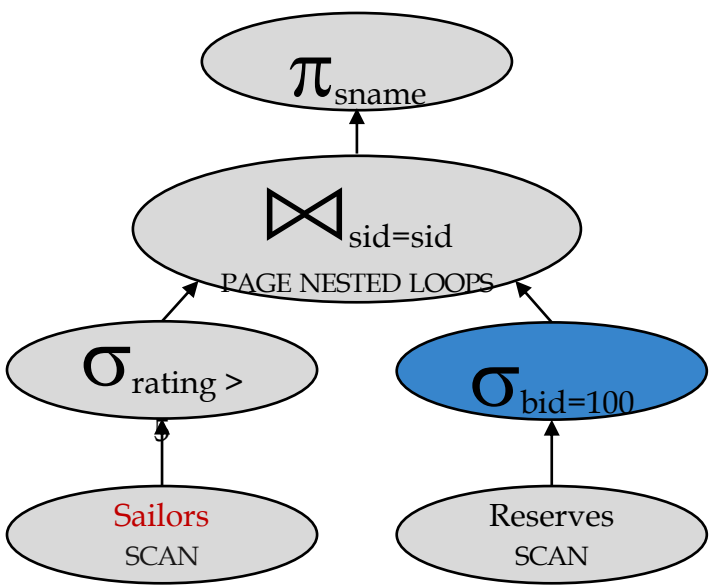
Plan3: Join Ordering



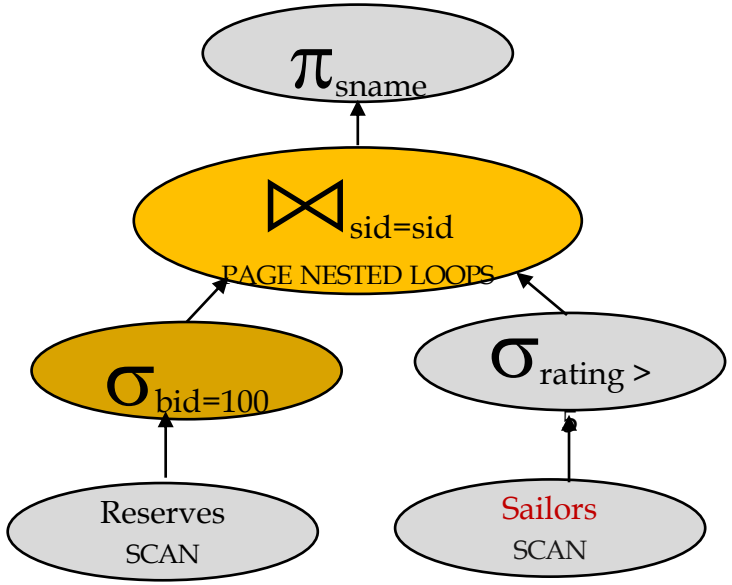


250,500 IOs

Plan 3 Cost



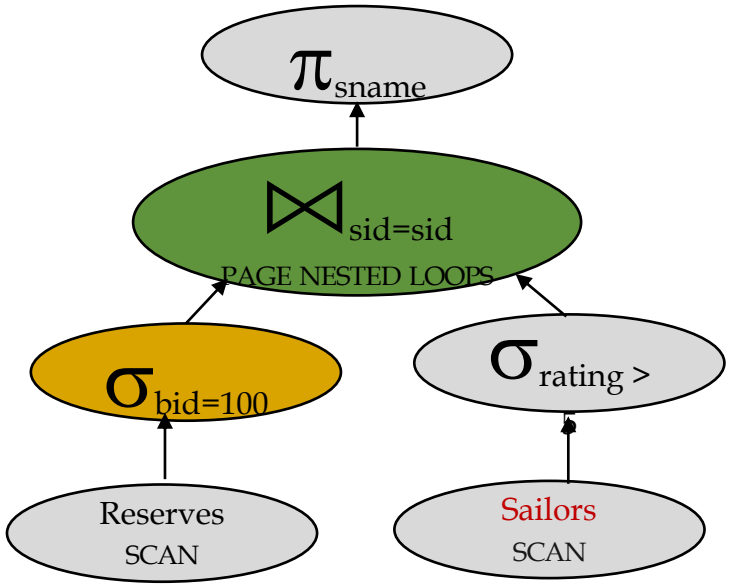
250,500 IOs



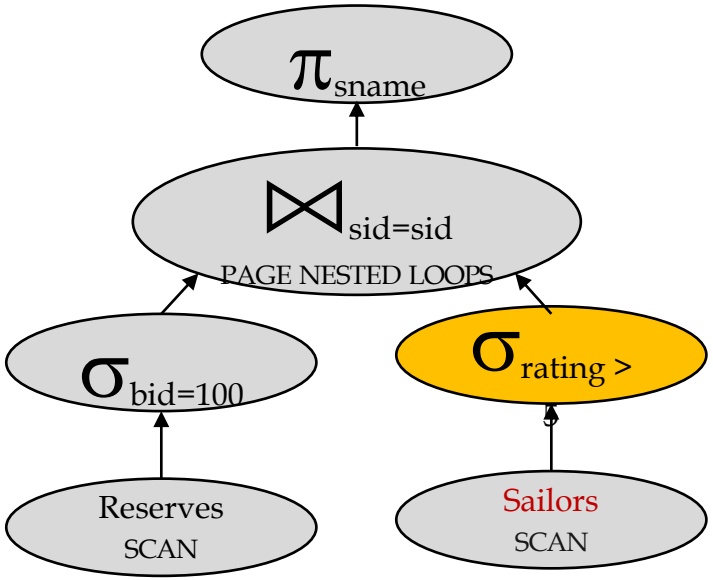
6000 IOs

- 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

Plan4: Materializing Inner Loops

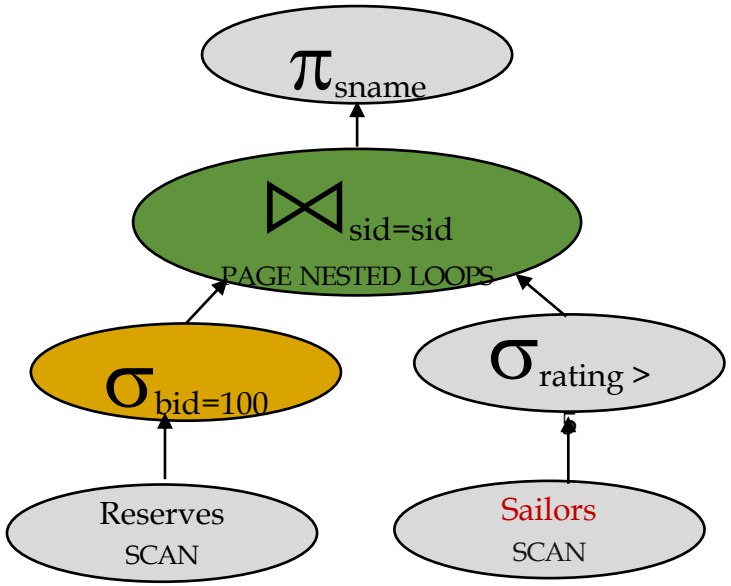


6000 IOs

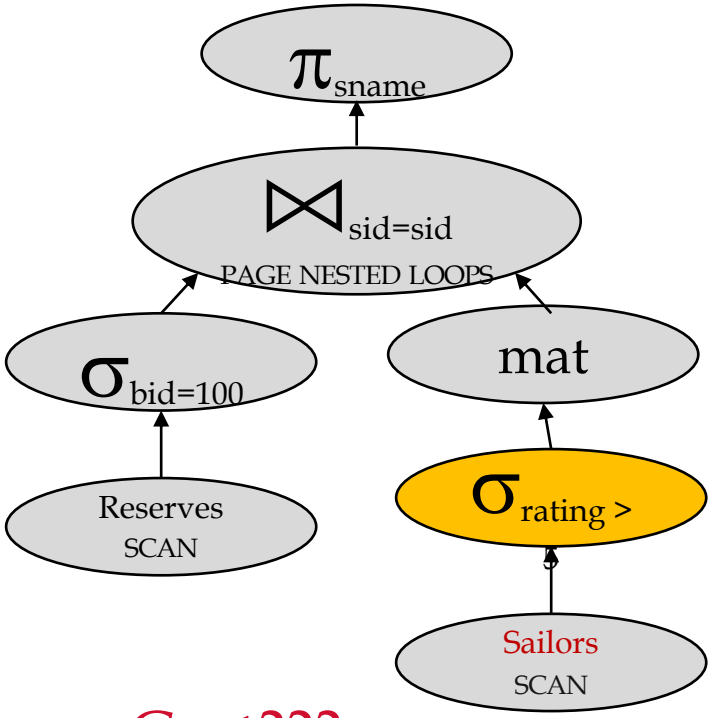


Cost???

Plan4: Materializing Inner Loops, Cost



6000 IOs

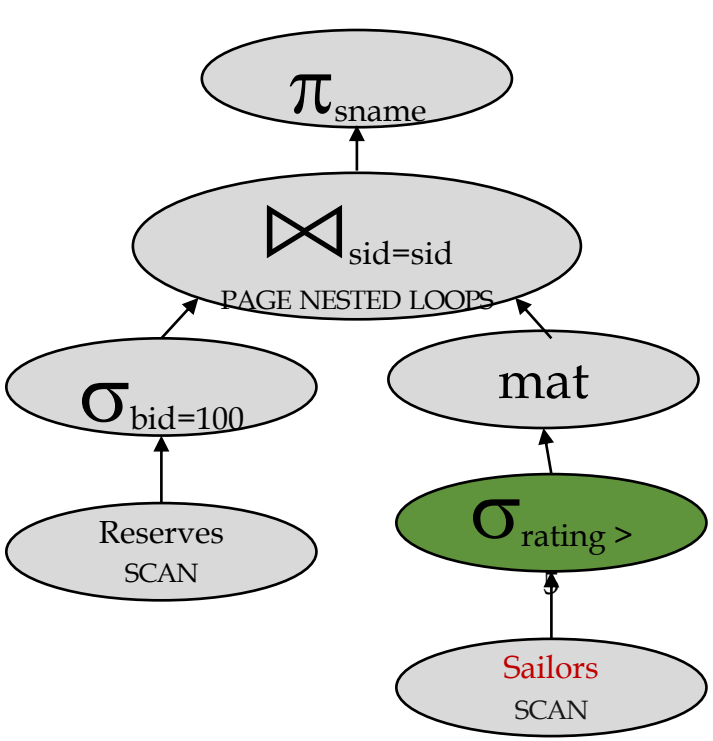


Cost???

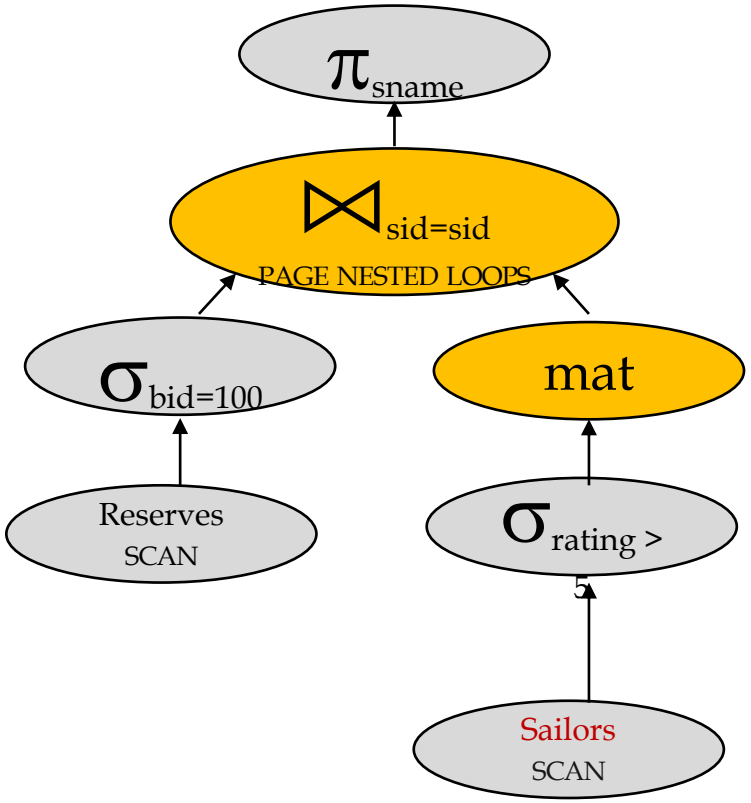
4250 IOs

- 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)

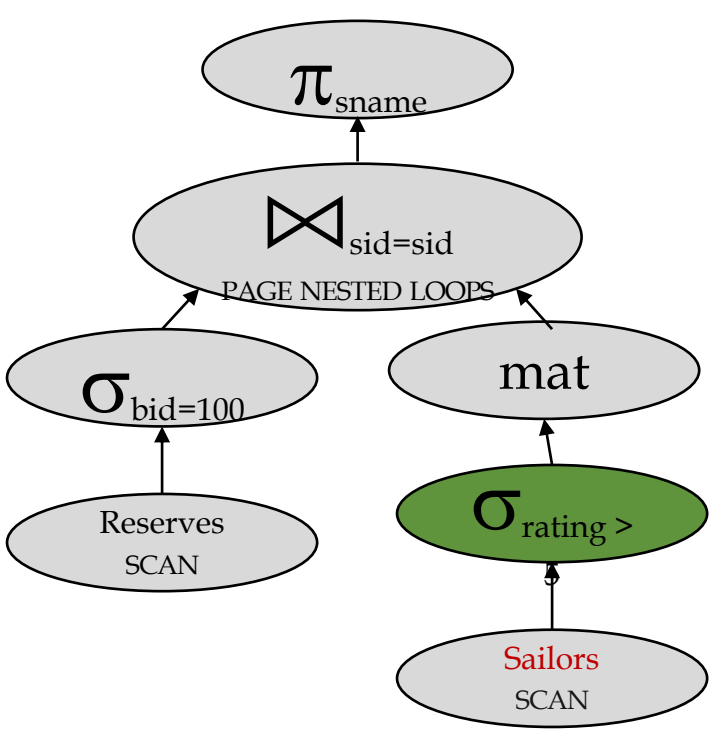
Plan5: Join Ordering Again



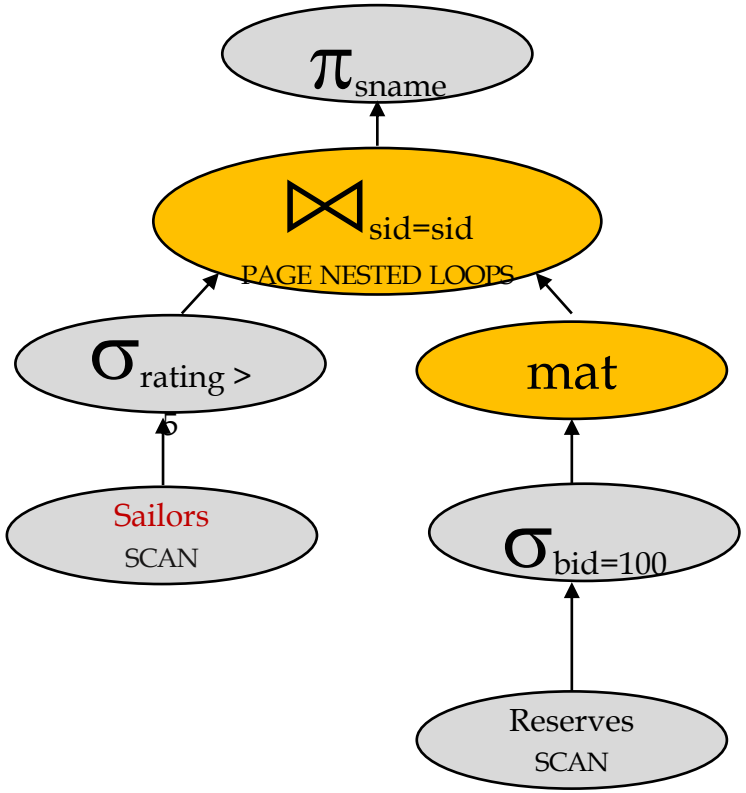
4250 IOs



Plan5: Join Ordering Again, Cost



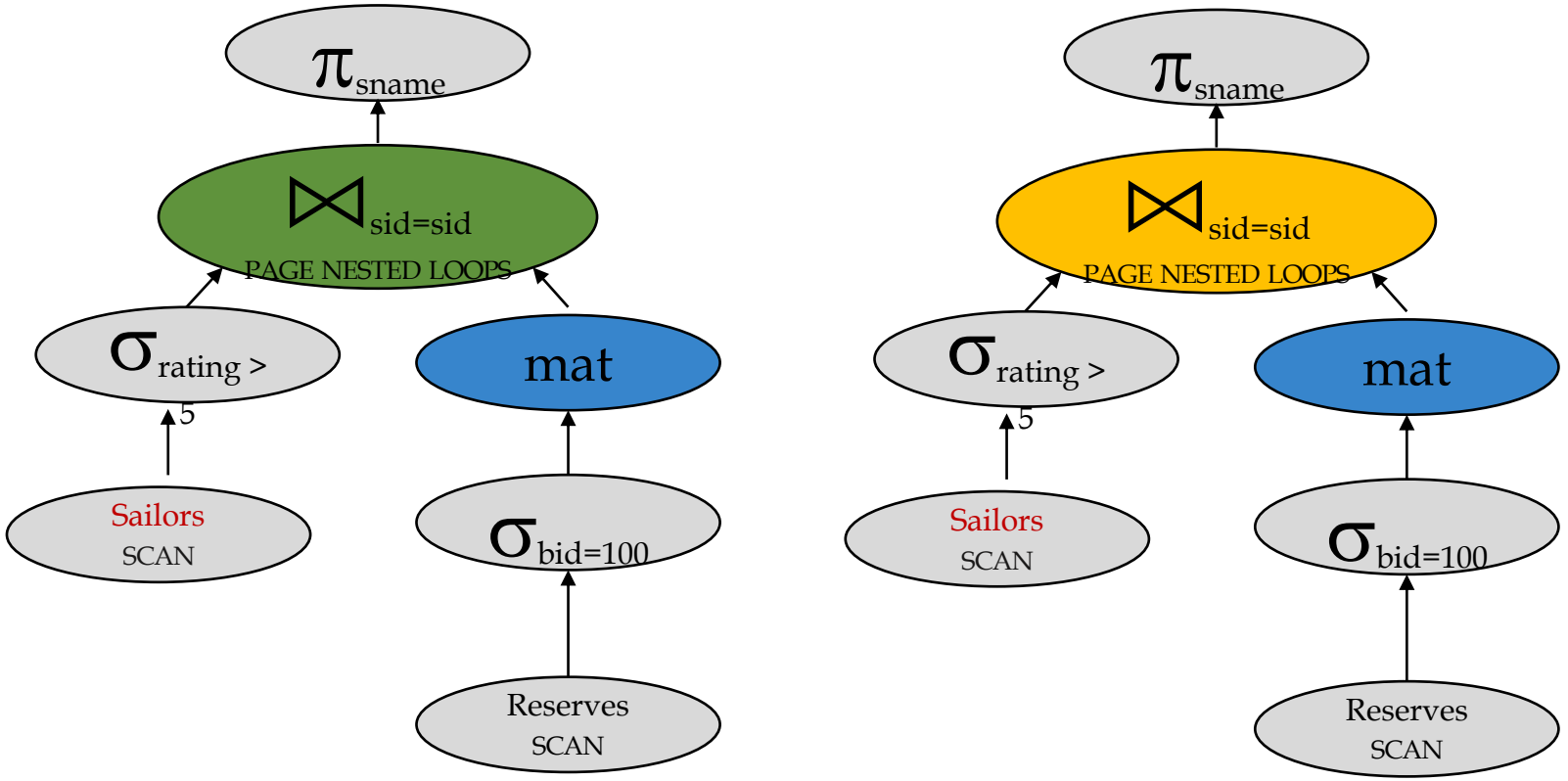
4250 IOs



Cost???
4010 IOs

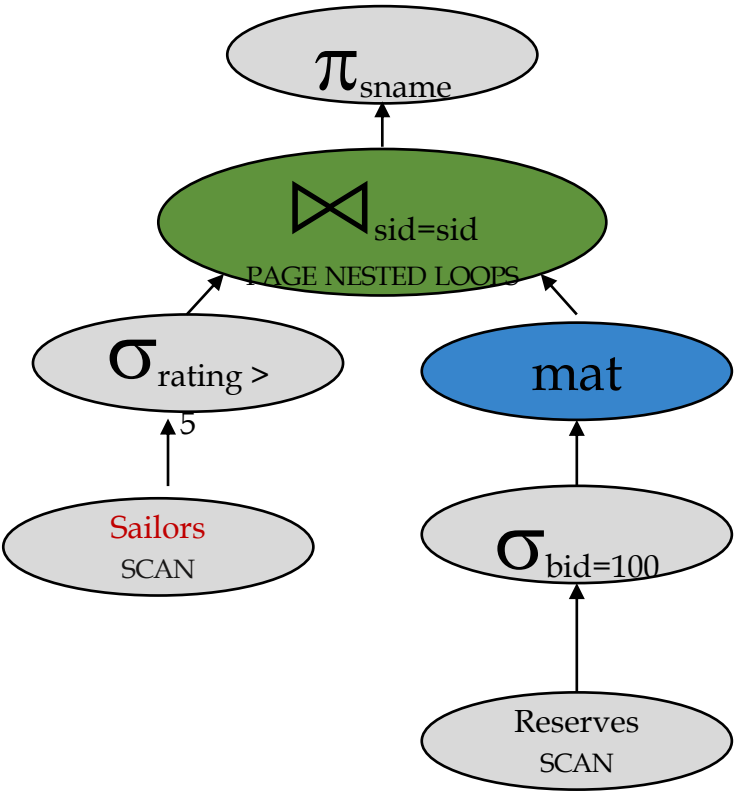
- Let's estimate the cost:
- Scan Sailors (500 IOs)
- Scan Reserves (1000 IOs)
- Materialize Temp table T1 (??? IOs)
- For each pageful of high-rated Sailors,
  Scan T1 (??? IOs)
- Total: 500 + 1000 + ??? + 250*???
- 500 + 1000 +10 +(250 *10)

Plan6: Join Algorithm

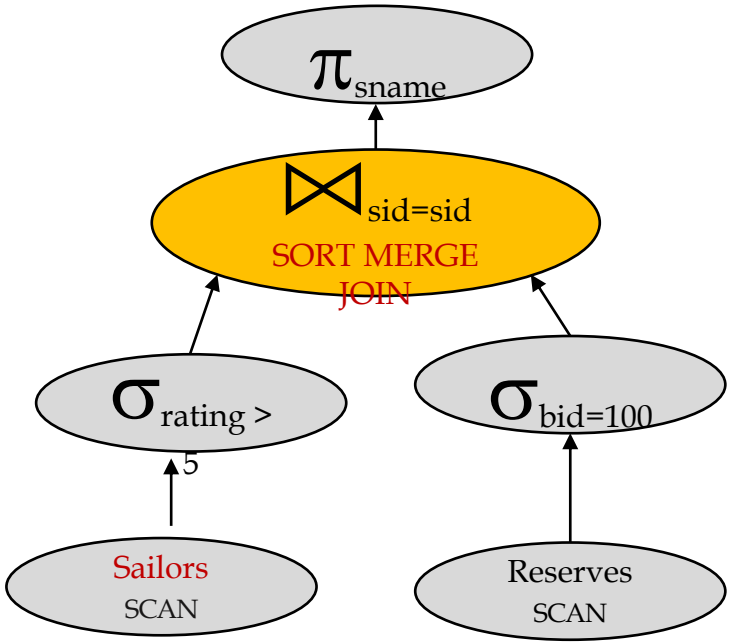


4010 IOs

Plan6: Join Algorithm, cost



4010 IOs



Cost???

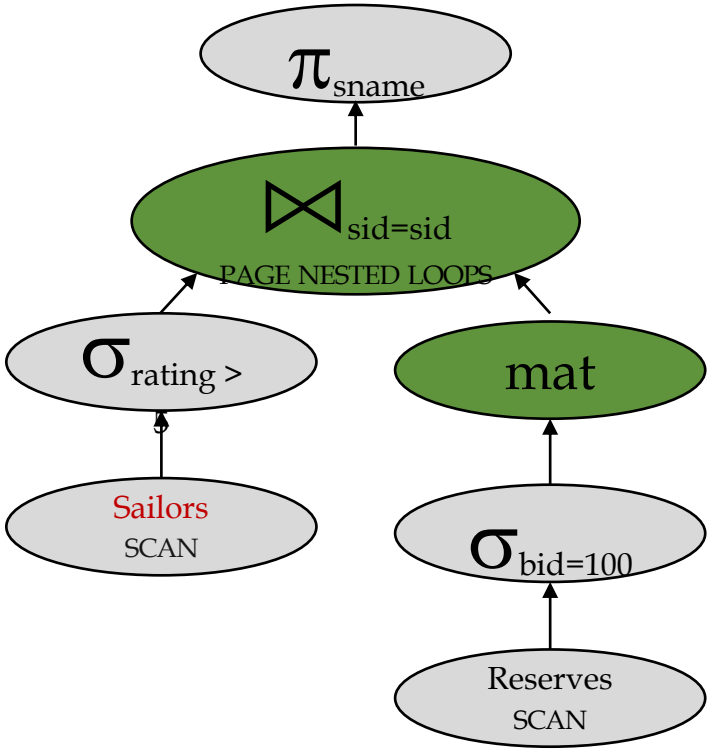
- With 5 buffers, cost of plan:
- Scan Reserves (1000)
- Scan Sailors (500)
- Sort high-rated sailors (???)

  250 + 3*2*250
- Sort reservations for boat 100 (???)

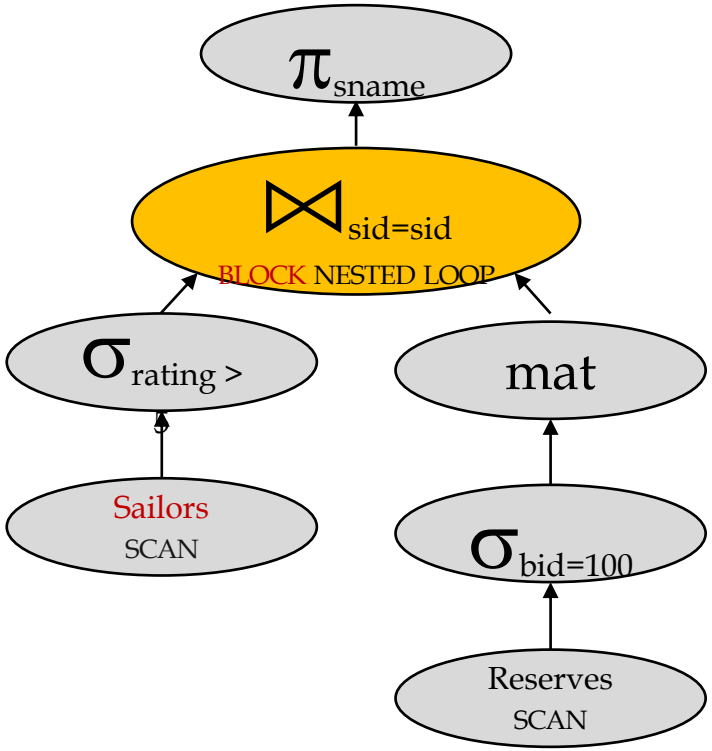
  10 + 2*10
- Merge (10+250) = 260
- Total:

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

Plan7: Join Algorithm Again, Again

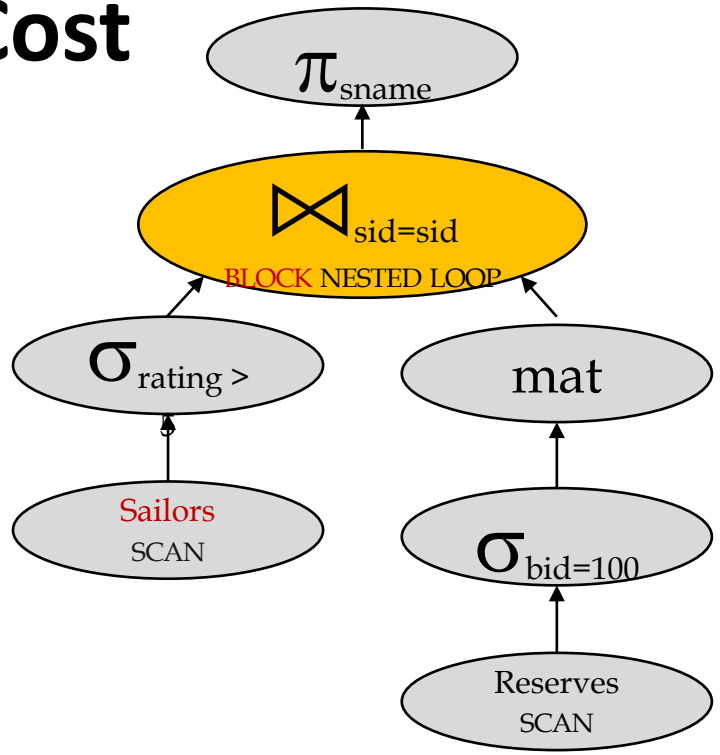


4010 IOs



Cost???

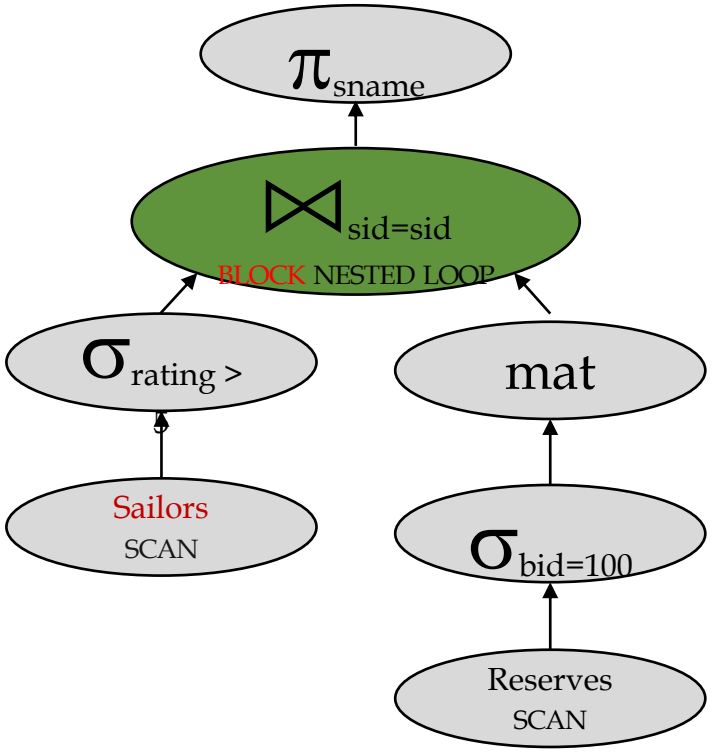
Plan7: Join Algorithm Again, Again. Cost

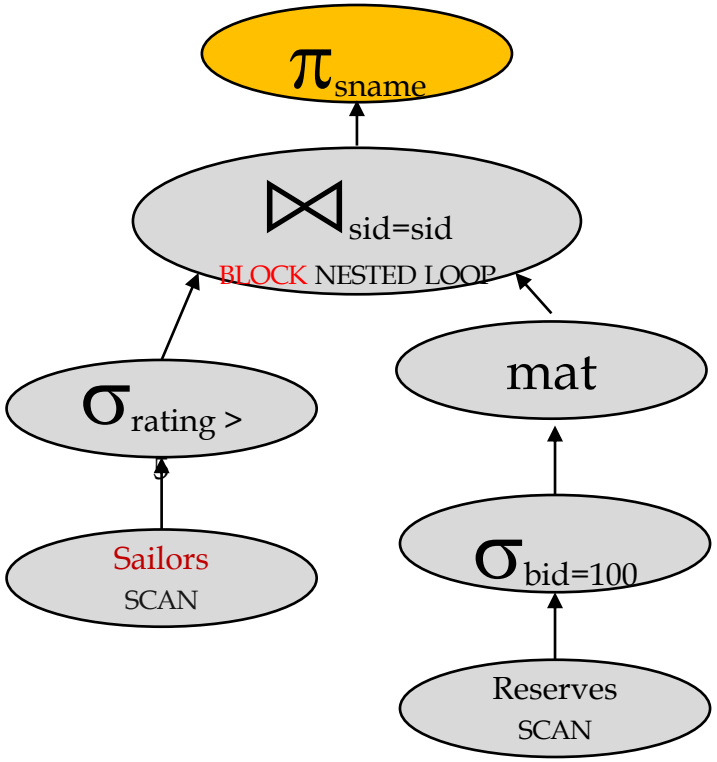


- With 5 buffers, cost of plan:
- Scan Sailors (500)
- Scan Reserves (1000)
- Write Temp T1 (10)
- For each ***blockful*** of high-rated sailors
- Loop on T1 (??? * 10)
- Total:

500 + 1000 +10 +(ceil(250/3) *10) = 500 + 1000 +10 +(84 *10) = 2350 IOs

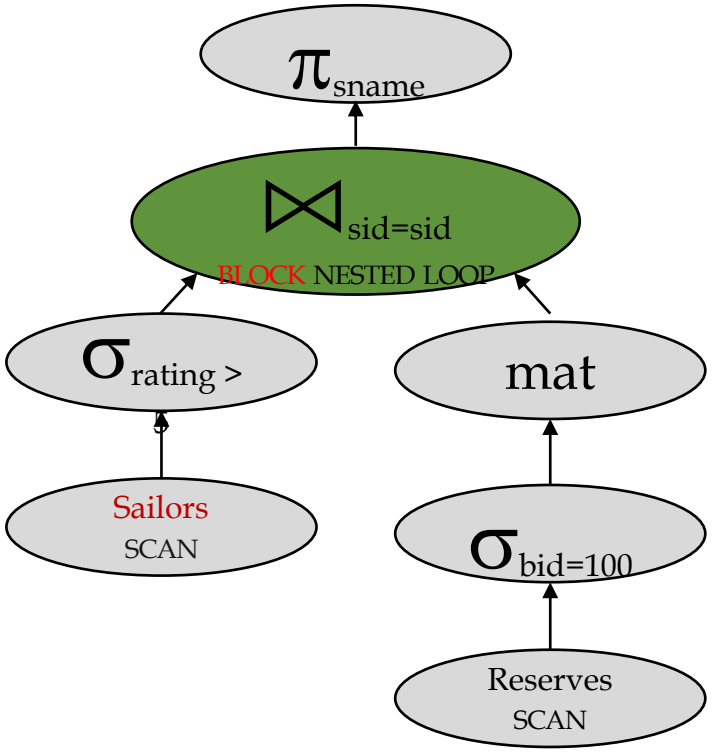
Projection Cascade & Pushdown



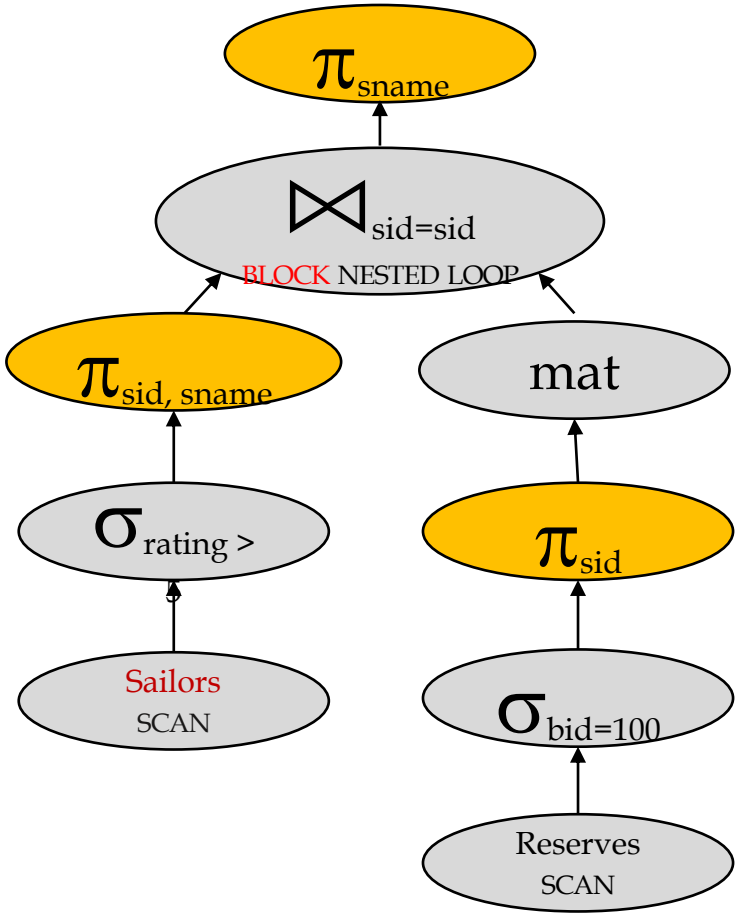


2350 IOs

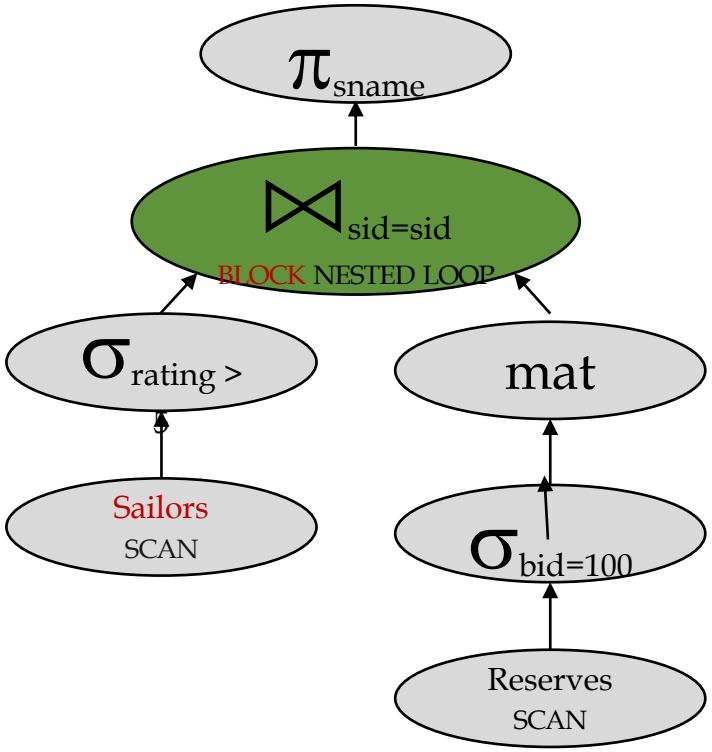
Projection Cascade & Pushdown, cont



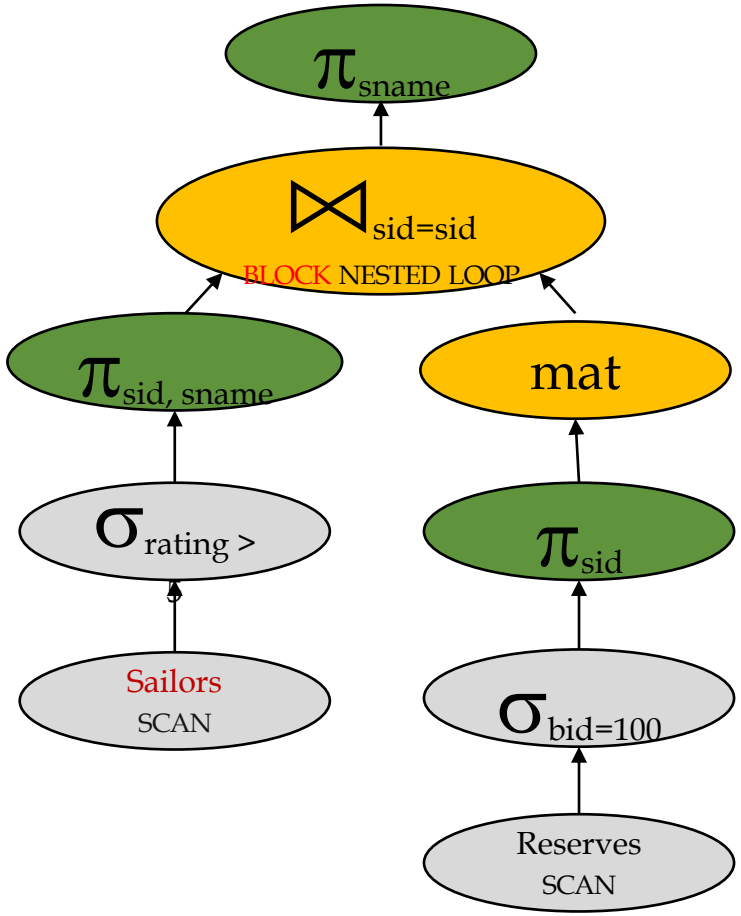
2350 IOs

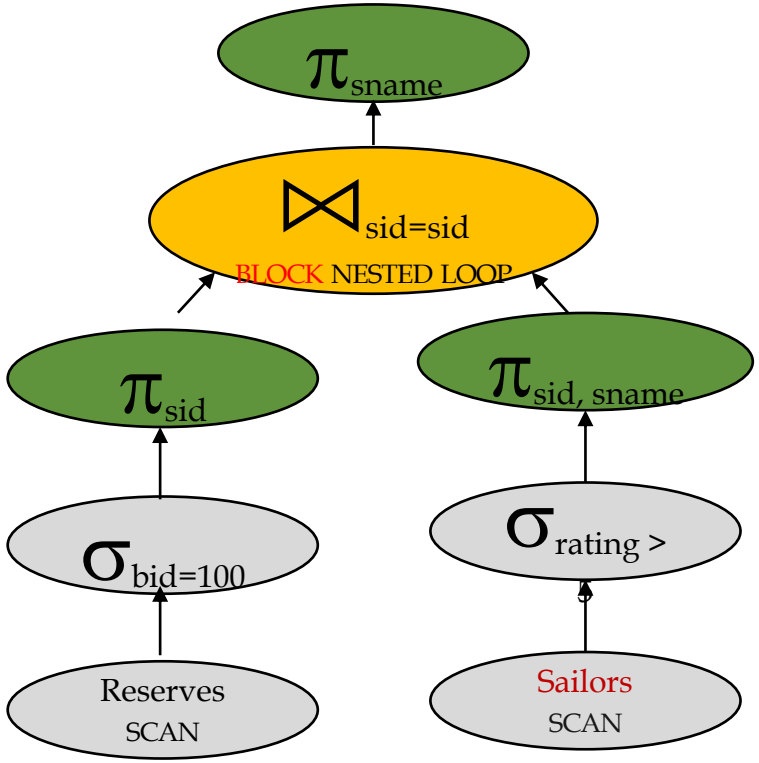


Plan 8: With Join Reordering, no Mat



2350 IOs



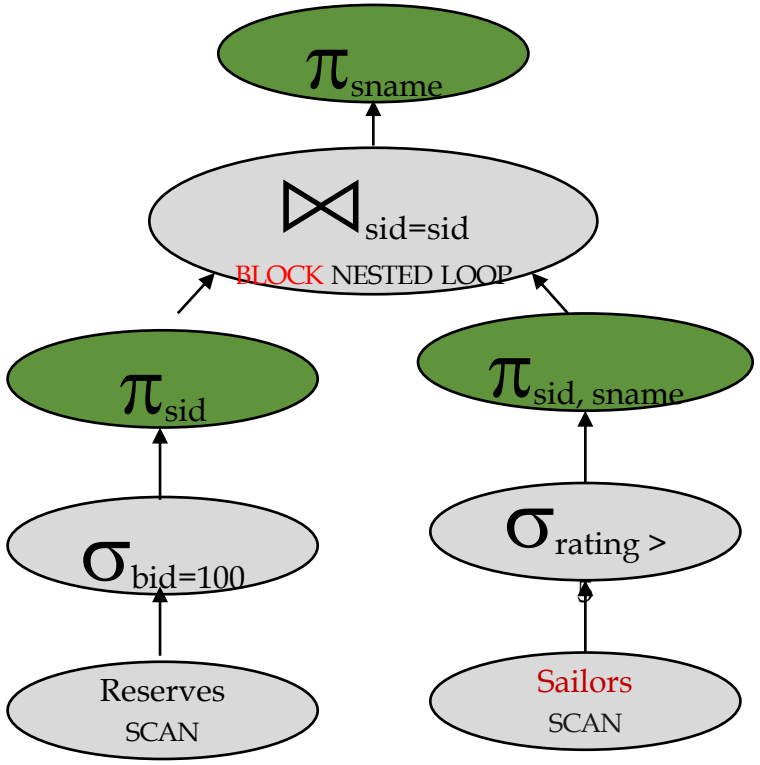


Plan 8: Cost Analysis

- 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)

  (1000/100)/10 = 1!

- Total: 1500

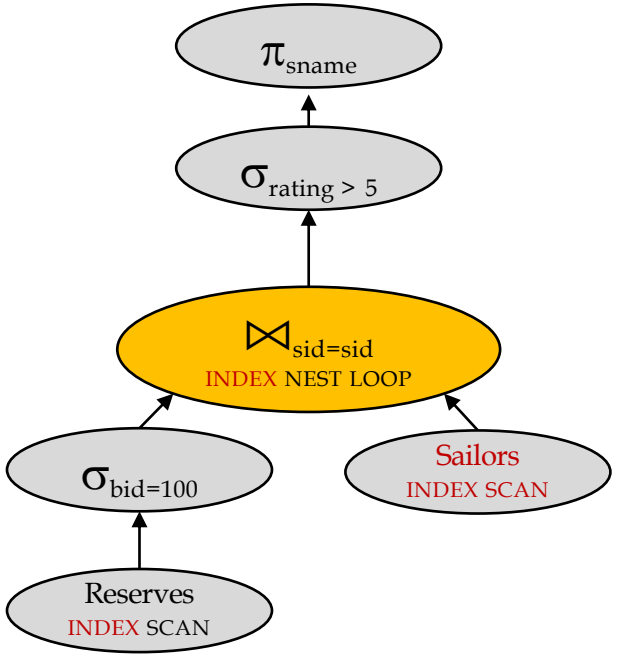


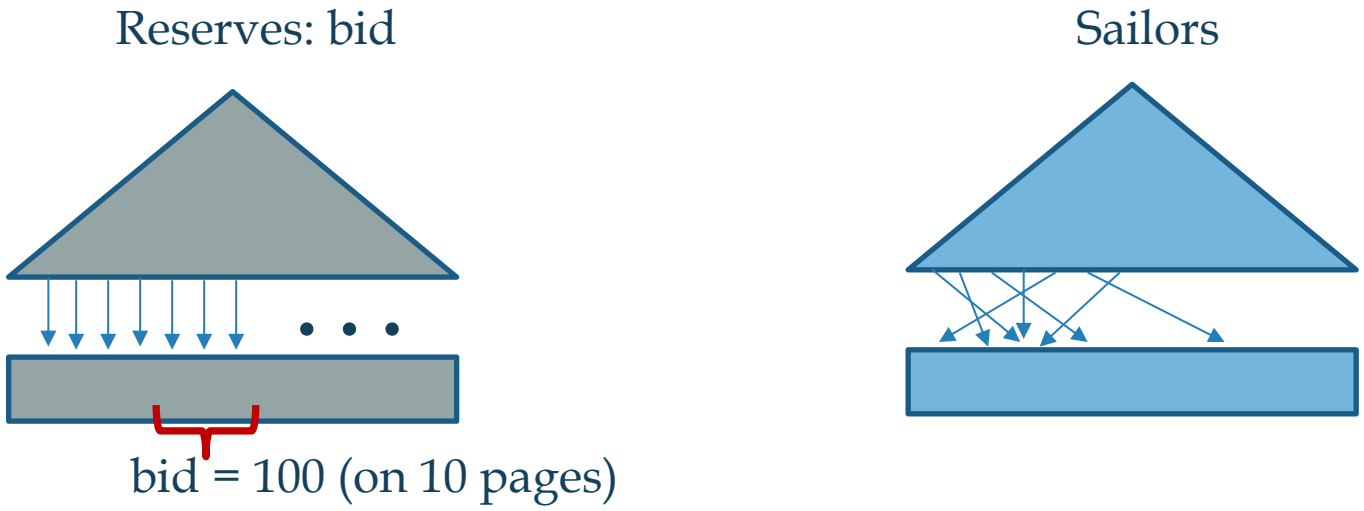
Plan 9*: Hash join

- **Hash join:** because the join attributes of R and S have the same domain, R and S can be hashed into the same hash file using the same hash function, then R ⋈ S can be computed based on the hash file.

How About Indexes?

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

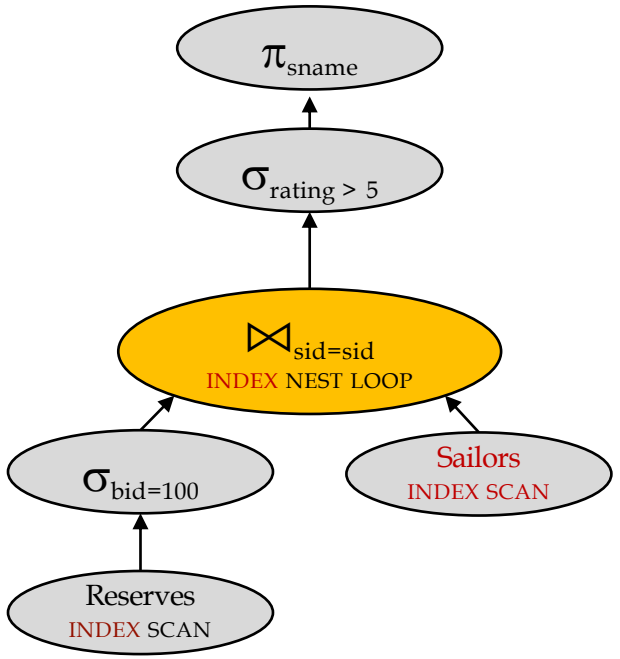




Index Cost Analysis Part 2

- 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

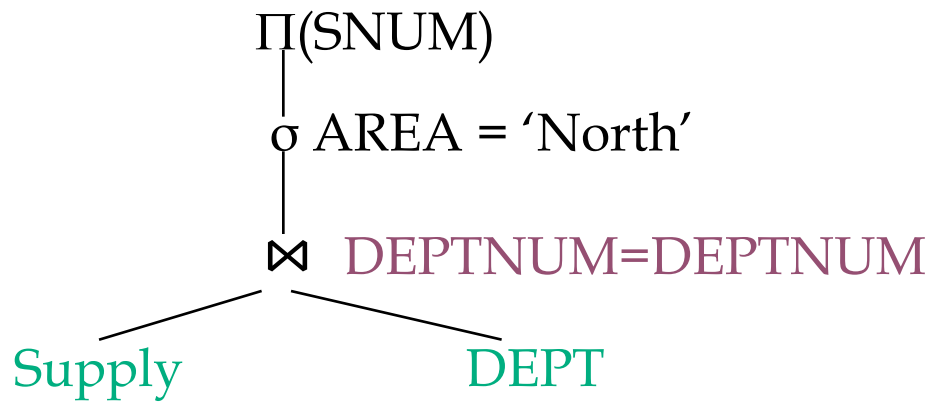
- There are *lots* of plans
  - Even for a relatively simple query
- Engineers often think they can pick good ones
  - E.g. MapReduce API was based on that assumption
- Not so clear that's true!
  - Manual query planning can be tedious, technical
  - Machines are better at enumerating options than people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

4.4.2 The Equivalent Transform of a Query

That is so called algebra optimization. It takes a series of transform on original query expression, and transform it into an equivalent, most effective form to be executed.

(1) Query tree

For example: $\Pi_{SNUM}\sigma_{AREA='NORTH'}(SUPPLY \bowtie_{DEPTNUM} DEPT)$



Leaves: relations
Middle nodes: unary/binary operations
Leaves → root: the executing order of operations

(2) The equivalent transform rules of relational algebra

1) Exchange rule of ⋈/×: $E1 \times E2 \equiv E2 \times E1$
2) Combination rule of ⋈/×: $E1 \times (E2 \times E3) \equiv (E1 \times E2) \times E3$
3) Cluster rule of Π: $\Pi_{A1\ldots An}(\Pi_{B1\ldots Bm}(E)) \equiv \Pi_{A1\ldots An}(E)$, legal when $A_1 \ldots A_n$ is the sub set of $\{B_1 \ldots B_m\}$
4) Cluster rule of σ: $\sigma_{F1}(\sigma_{F2}(E)) \equiv \sigma_{F1 \wedge F2}(E)$
5) Exchange rule of σ and Π: $\sigma_F(\Pi_{A1\ldots An}(E)) \equiv \Pi_{A1\ldots An}(\sigma_F(E))$ if F includes attributes $B_1 \ldots B_m$ which don't belong to $A_1 \ldots A_n$, then $\Pi_{A1\ldots An}(\sigma_F(E)) \equiv \Pi_{A1\ldots An}\, \sigma_F(\Pi_{A1\ldots An\ ,\ B1\ldots Bm}(E))$
6) If the attributes in F are all the attributes in E1, then

   $\sigma_F(E1 \times E2) \equiv \sigma_F(E1) \times E2$

if F in the form of F1∧F2, and there are only E1's attributes in F1, and there are only E2's attributes in F2, then $\sigma_F(E1 \times E2) \equiv \sigma_{F1}(E1) \times \sigma_{F2}(E2)$

if F in the form of F1∧F2, and there are only E1's attributes in F1, while F2 includes the attributes both in E1 and E2, then $\sigma_F(E1 \times E2) \equiv \sigma_{F2}(\sigma_{F1}(E1) \times E2)$

7) $\sigma_F(E1 \cup E2) \equiv \sigma_F(E1) \cup \sigma_F(E2)$
8) $\sigma_F(E1 - E2) \equiv \sigma_F(E1) - \sigma_F(E2)$
9) Suppose $A_1 \ldots A_n$ is a set of attributes, in which $B_1 \ldots B_m$ are E1's attributes, and $C_1 \ldots C_k$ are E2's attributes, then

   $\Pi_{A1 \ldots An}(E1 \times E2) \equiv \Pi_{B1 \ldots Bm}(E1) \times \Pi_{C1 \ldots Ck}(E2)$
10) $\Pi_{A1 \ldots An}(E1 \cup E2) \equiv \Pi_{A1 \ldots An}(E1) \cup \Pi_{A1 \ldots An}(E2)$

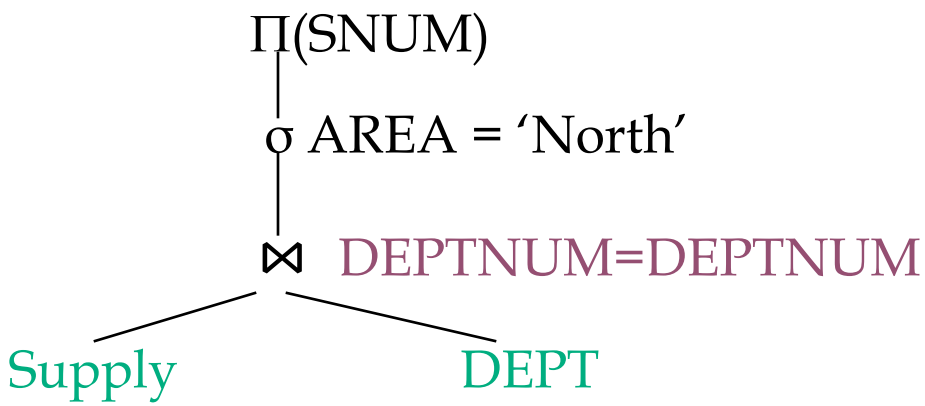
Basic principles

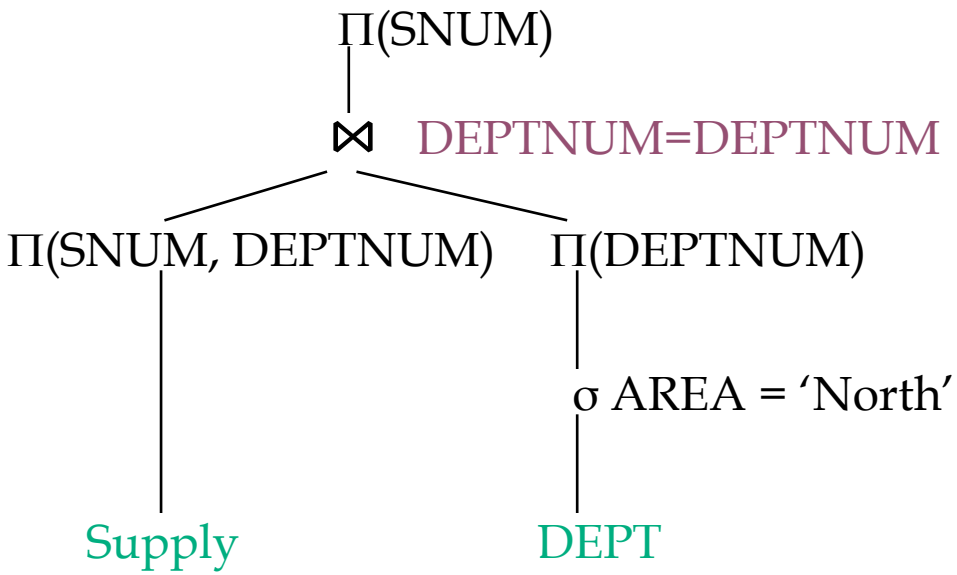
The target of algebra optimization is to make the scale of the operands which involved in binary operations be as small as possible :

- ✓ Push down the unary operations as low as possible
- ✓ Look for and combine the common sub-expression

Example

$$\Pi_{SNUM}\sigma_{AREA='NORTH'}(SUPPLY \bowtie_{DEPTNUM} DEPT)$$





4.4.3 The Operation Optimization

How to find a “good” access strategy to compute the query improved by algebra optimization is introduced in this section:

- ***Optimization of select operation***
- ***Optimization of project operation***
- ***Optimization of join operation***
- *Optimization of combined operations*
- *Optimization of set operation*

Optimization of join operation

Nested loop: one relation acts as outer loop relation (O), the other acts as inner loop relation (I). For every tuple in O, scan I one time to check join condition.
Because the relation is accessed from disk in the unit of block, we can use ***block buffer*** to improve efficiency.

For $R \bowtie S$, if let R as O, S as I, b_R is physical block number of R, b_S is physical block number of S, there are n_B block buffers in system (n_B>=2), and n_B-1 buffers used for O, one buffer used for I, then the total disk access times needed to compute $R \bowtie S$ is: $b_R + \lceil b_R/(n_B-1) \rceil \times b_S$

Optimization of join operation

- **Merge scan**: order the relation R and S on disk in ahead, then we can compare their tuples in order, and both relation only need to scan one time. ***If*** R and S have not ordered in ahead, must consider the ordering cost to see if it is worth to use this method.
- **Using index or hash** to look for mapping tuples: in nested loop method, if there is suitable access route on I (say B+ tree index), it can be used to substitute sequence scan. It is best when there is cluster index or hash on join attributes.
- **Hash join:** because the join attributes of R and S have the same domain, R and S can be hashed into the same hash file using the same hash function, then R ⋈ S can be computed based on the hash file. It is suitable for the case that two tables need to be joined frequently.

What is needed for query optimization?

- **Given: A closed set of operators**
  - Relational ops (table in, table out)
  - Physical implementations (of those ops and a few more)

1.Plan space

  - Based on relational equivalences, different implementations

2.Cost Estimation based on

  - Cost formulas
  - Size estimation, in turn based on
    - Catalog information on base tables
    - Selectivity (Reduction Factor) estimation
- Target: To sift through the plan space and find lowest cost option!

Plan Space Review

- For a SQL query, full plan space:
  - All equivalent relational algebra expressions
    - Based on the equivalence rules we learned
  - All mixes of physical implementations of those algebra expressions
- We might prune this space:
  - Selection/Projection pushdown
  - Avoid cartesian products

Plan Space: Too Large, must be pruned.

- Only the space of ***left-deep plans*** is considered.
- Left-deep plans allow output of each operator to be ***pipelined*** into the next operator without storing it in a temporary relation.
- Cartesian products avoided.

Cost Estimation

- For each plan considered, must estimate total cost:
  - Must estimate ***cost*** of each operation in plan tree.
    - Depends on input cardinalities.
    - We've already discussed this for various operators
      - sequential scan, index scan, joins, etc.
  - Must estimate ***size of result*** for each operation in tree!
    - Because it determines downstream input cardinalities!
    - Use information about the input relations.
    - For selections and joins, assume independence of predicates.

Statistics and Catalogs

- Need info on relations and indexes involved.
- **Catalogs** typically contain at least:

Statistic	Meaning
NTuples	# of tuples in a table (cardinality)
NPages	# of disk pages in a table
Low/High	min/max value in a column
Nkeys	# of distinct values in a column
IHeight	the height of an index
INPages	# of disk pages in an index

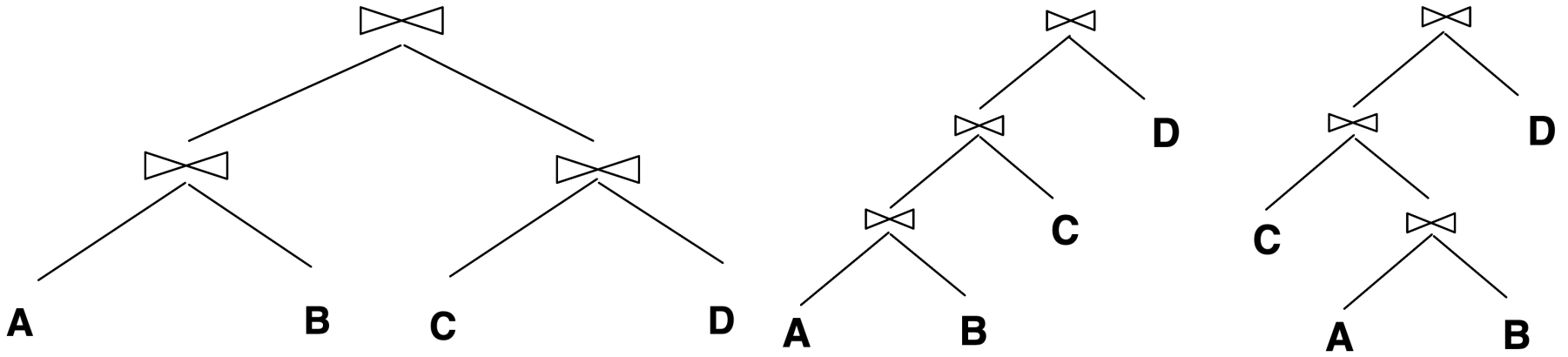
- Catalogs updated periodically.
  - Too expensive to do continuously
  - Lots of approximation anyway, so a little slop here is ok.
- Modern systems do more
  - Esp. keep more detailed statistical information on data values
    - e.g., ***histograms***

Cost Estimates for Single-Relation Plans

- Index I on primary key matches selection:
  - *Cost is Height(I)+1 for a B+ tree, about 1.2 for hash index.*
- Clustered index I matching one or more selects:
  - *(NPages(I)+NPages(R)) * product of RF's of matching selects.*
- Non-clustered index I matching one or more selects:
  - *(NPages(I)+NTuples(R)) * product of RF's of matching selects.*
- Sequential scan of file:
  - *NPages(R).*
- + ***Note:*** *Typically, no duplicate elimination on projections! (Exception: Done on answers if user says DISTINCT.)*

Queries Over Multiple Relations

- Fundamental decision in System R: *only* ***left-deep join trees*** are considered.
- As the number of joins increases, the number of alternative plans grows rapidly; ***we need to restrict the search space.*** Left-deep trees allow us to generate all *fully pipelined* plans.
- Intermediate results not written to temporary files.
  Not all left-deep trees are fully pipelined (e.g., SortMerge join).



Enumeration of Left-Deep Plans

- Left-deep plans differ only in the order of relations, the access method for each relation, and the join method for each join.
- Enumerated using N passes (if N relations joined):
  - Pass 1: Find best 1-relation plan for each relation.
  - Pass 2: Find best way to join result of each 1-relation plan (as outer) to another relation. *(All 2-relation plans.)*
  - Pass N: Find best way to join result of a (N-1)-relation plan (as outer) to the N'th relation. *(All N-relation plans.)*
- For each subset of relations, retain only:
  - Cheapest plan overall, plus
  - Cheapest plan for each *interesting order* of the tuples.

Enumeration of Plans (Contd.)

- ORDER BY, GROUP BY, aggregates etc. handled as a final step, using either an `interestingly ordered' plan or an additional sorting operator.

- An N-1 way plan is not combined with an additional relation unless there is a join condition between them, unless all predicates in WHERE have been used up.
  - i.e., avoid Cartesian products if possible.

- In spite of pruning plan space, this approach is still exponential in the # of tables.

Cost Estimation for Multi-relation Plans

- Consider a query block:
- Maximum # tuples in result is the product of the cardinalities of relations in the FROM clause.
- ***Reduction factor (RF)*** associated with each *term* reflects the impact of the *term* in reducing result size. *Result cardinality* = Max # tuples * product of all RF's.
- Multi-relation plans are built up by joining one new relation at a time.
  - Cost of join method, plus estimation of join cardinality gives us both cost estimate and result size estimate