THE UNIVERSITY of EDINBURGH

# Advanced Database Systems

Spring 2024

Lecture #21:

## Query Optimisation: Plan Space Example

R&G: Chapter 15

1



# The Plan Space of a Simple Query

2



# Example Database

**Reserves**

| sid | bid | day | rname |
|---|---|---|---|
| | | | |
| | | | |

1000 pages, 100 tuples per page

Each tuple is 40 bytes long

Assume 100 boats (each equally likely)

**Sailors**

| sid | sname | rating | age |
|---|---|---|---|
| | | | |
| | | | |

500 pages, 80 tuples per page

Each tuple is 50 bytes long

Assume 10 different ratings (each equally likely)

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

Remember: just counting I/Os

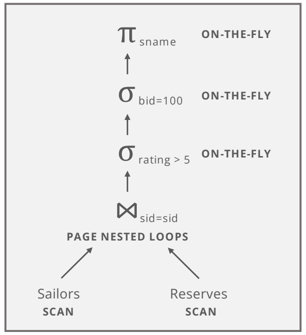
3



# Query Plan 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 ⇒




4

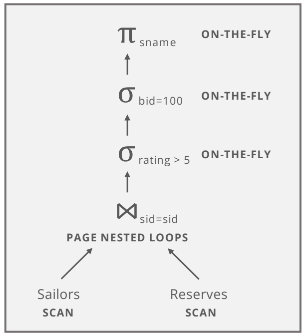
# Query Plan 1 Cost

Cost estimation:

Scan Sailors: **500 I/Os**

For each page of Sailors
Scan Reserves: **1000 I/Os**

Total = 500 + 500 · 1000
= 500,500 I/Os








# Query Plan 1 Cost Analysis

Cost: 500,500 I/Os

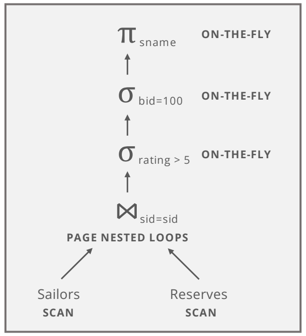
By no means a terrible plan!

Misses several opportunities

- Selections could be 'pushed' down
- No use of indexes

Goal of optimisation

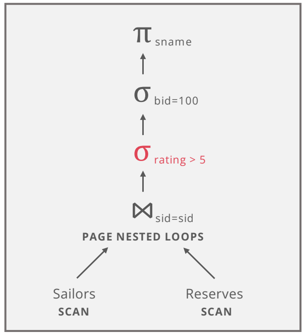
- Find faster plans that compute the same answer



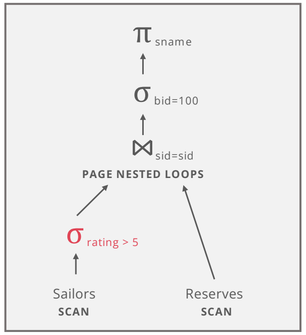





# Selection Pushdown




500,500 I/Os




Cost?





# Query Plan 2 Cost

Cost estimation:

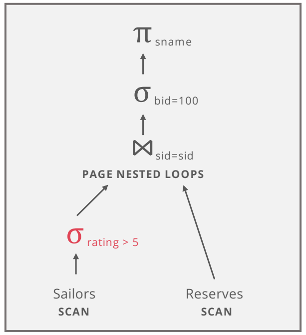
Scan Sailors: **500 I/Os**

For each page of high-rated Sailors
Scan Reserves: **1000 I/Os**

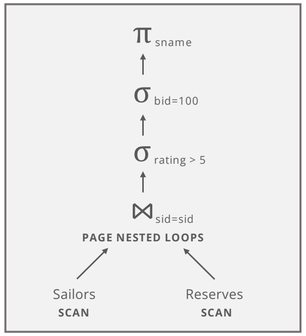
Total = 500 + ??? · 1000

Remember: 10 ratings, all equally likely

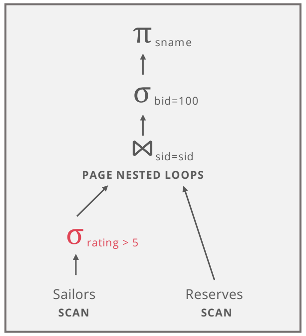
Total = 500 + (500 / 2) · 1000
= 250,500 I/Os



# Decision 1




500,500 I/Os

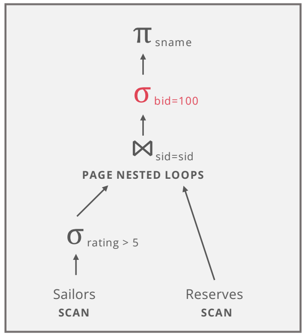



250,500 I/Os

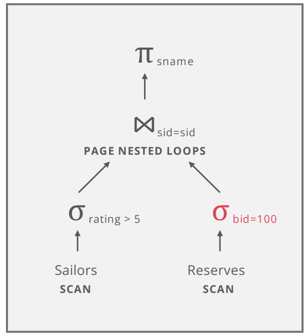




# More Selection Pushdown




250,500 I/Os




Cost?





# Query Plan 3 Cost

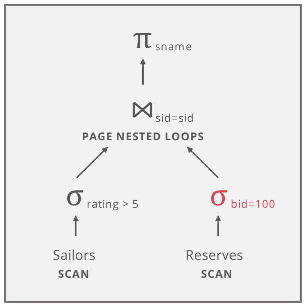
Cost estimation:

- Scan Sailors: **500 I/Os**
- For each page of high-rated Sailors
  - Read through Reserves tuples that match
- Total = 500 + 250 · ???

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

Problem: This does **not** actually save any I/Os

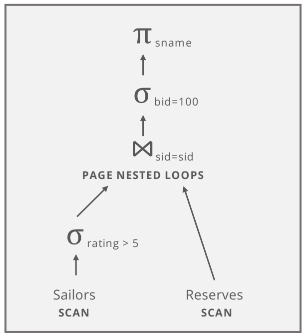
- To find matching Reserves tuples, we end up scanning Reserves the same # of times (1000)



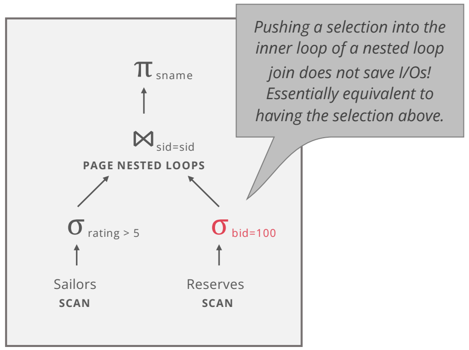





# Decision 2




250,500 I/Os




250,500 I/Os

# So Far, We've Tried

Basic page nested loops (500,500)

Selection pushdown on left (250,500)

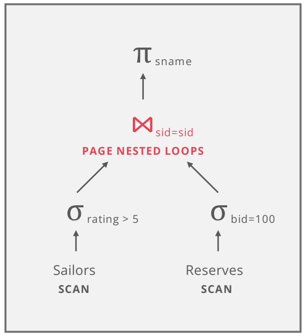
More selection pushdown on right (250,500)

Next: join ordering

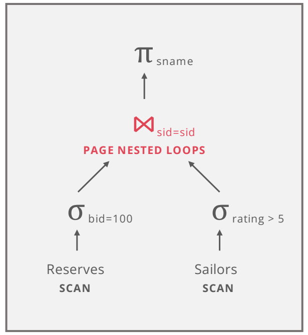




# Join Ordering




250,500 I/Os




Cost?





# Query Plan 4 Cost

Cost estimation:

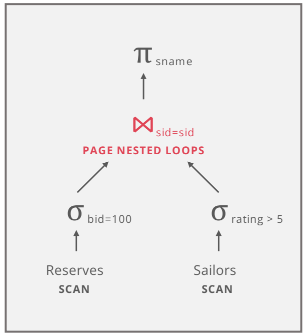
Scan Reserves: **1000 I/Os**

For each page of Reserves for bid 100
Scan Sailors: **500 I/Os**

Total = 1000 + ??? · 500

Uniformly distributed across 100 boat values

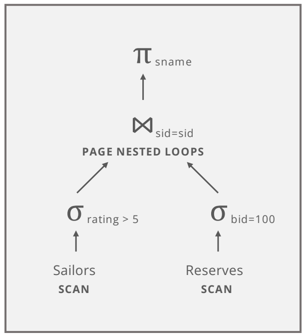
Total = 1000 + (1000 / 100) · 500
= 6000 I/Os



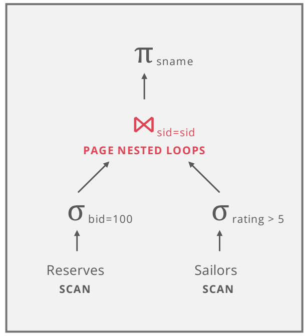





# Decision 3




250,500 I/Os




6000 I/Os

## 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: materialisation

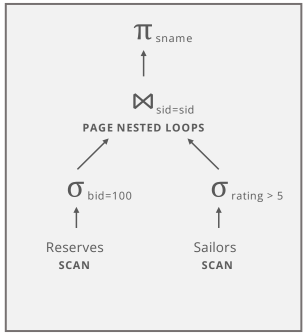




## Materialising Inner Loops

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?




6000 I/Os





## Query Plan 5 Cost

Cost estimation:

Scan Reserves: **1000 I/Os**

Scan Sailors: **500 I/Os**

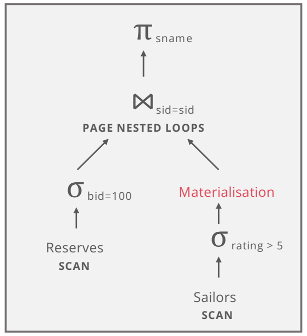
Materialise temp table T1: ??? **I/Os**

For each page of Reserves for bid 100
Scan T1: ??? **I/Os**

Total = 1000 + 500 + ??? + 10 · ???

Ratings from 1 to 10, uniformly distributed

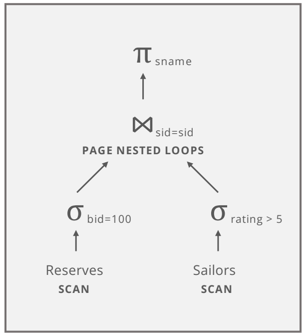
Total = 1000 + 500 + 250 + 10 · 250 = 4250 I/Os



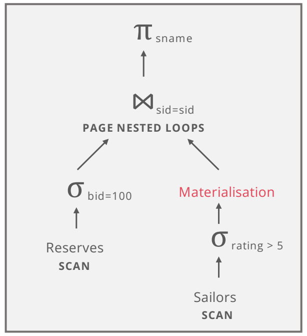





## Decision 4



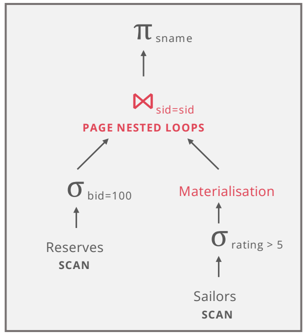

6000 I/Os




4250 I/Os

# Join Ordering Again

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




4250 I/Os





# Query Plan 6 Cost

Cost estimation:

Scan Sailors: **500 I/Os**

Scan Reserves: **1000 I/Os**

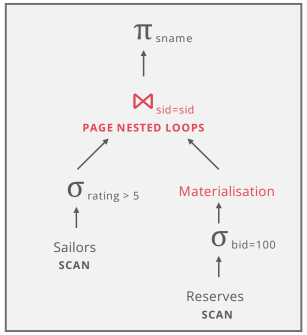
Materialise temp table T1: ??? **I/Os**

For each page of high-rated Sailors
Scan T1: ??? **I/Os**

Total = 500 + 1000 + ??? + 250 · ???

100 boat values, uniformly distributed

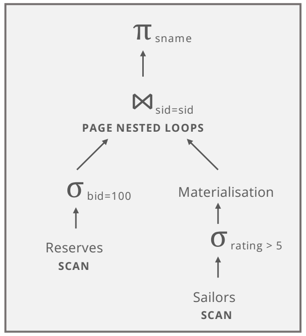
Total = 500 + 1000 + 10 + 250 · 10 = 4010 I/Os



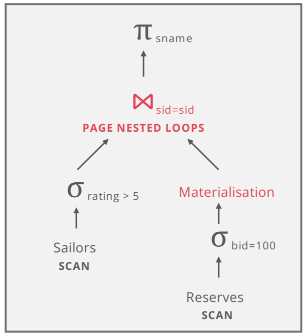





# Decision 5




4250 I/Os




4010 I/Os





# 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)

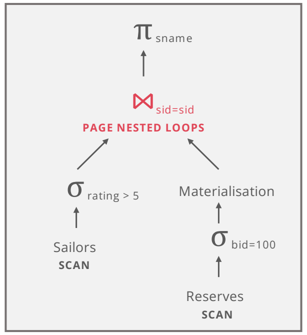
Materialising inner loop (4250)

Join ordering again with materialisation (4010)

Next: sort merge join

# Join Algorithm

What if change the join algorithm?








# Query Plan 7 Cost

Cost estimation with 5 buffers:

Scan Sailors: **500 I/Os**

Scan Reserves: **1000 I/Os**

Sort high-rated Sailors: ??? **I/Os**

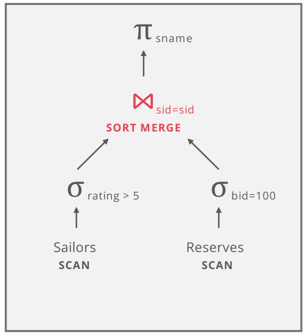
Pass 0 doesn't do read I/O, just gets input from select

Sort reservations for boat 100 : ??? **I/Os**

Pass 0 doesn't do read I/O, just gets input from select

How many passes for each sort?

Merge: (10 + 250) = **260 I/Os**








# Query Plan 7 Cost, Part 2

External sort with 5 buffers:

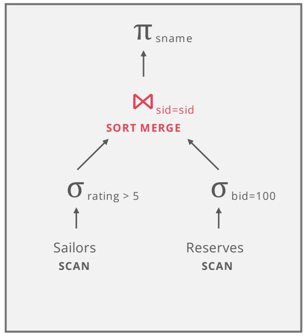
$1 + \lceil \log_4(10/5) \rceil = 2$ passes for Reserves

Pass 0 = **10** to write
Pass 1 = 2 · 20 = **20** to read/write

$1 + \lceil \log_4(250/5) \rceil = 4$ passes for Sailors

Pass 0 = **250** to write
Pass 1, 2, 3 = 2 · 250 = **500** to read/write

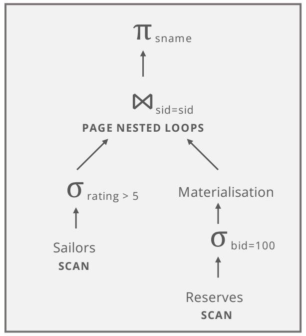
Total = scan both (1000 + 500) +
sort Reserves (10 + 20) +
sort Sailors (250 + 3 · 500) + merge (260) = 3540 I/Os



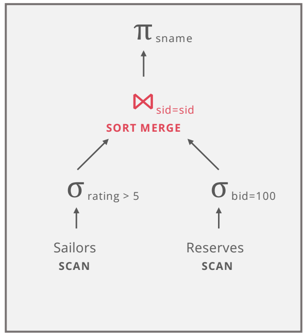





# Decision 6




4010 I/Os




3540 I/Os

# 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)

Materialising inner loop (4250)

Join ordering again with materialisation (4010)

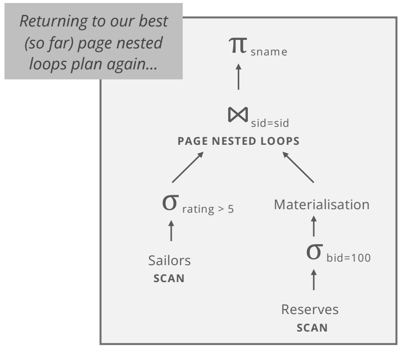
Sort merge join (3540)

Next: block nested loops join

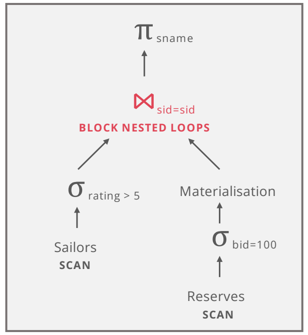




# Join Algorithm Again, Again




4010 I/Os
(and sort merge at 3510 I/Os)




Cost?





# Query Plan 8 Cost

Cost estimation with 5 buffers:

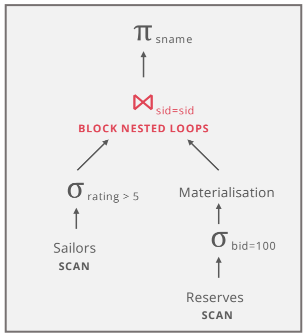
Scan Sailors: **500 I/Os**

Scan Reserves: **1000 I/Os**

Write temp table T1: **10 I/Os**

For each block of high-rated Sailors
Iterate over T1: ??? · **10 I/Os**

Block size = 3, #blocks (???) = ceil(250/3) = 84
Sailors tuples pipelined from select

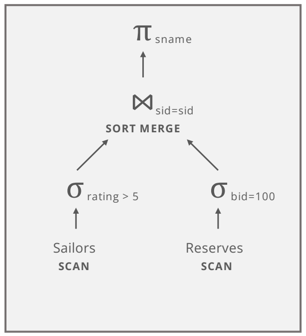



Total = scan both (500 + 1000) + write T1 (10) + BNLJ (84 · 10) = 2350 I/Os

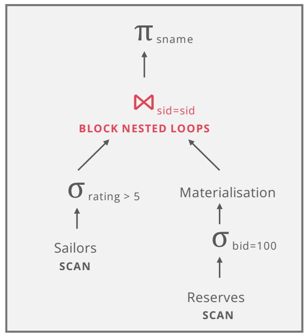




# Decision 7




3540 I/Os




2350 I/Os

# 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)

Materialising inner loop (4250)

Join ordering again with materialisation (4010)

Sort merge join (3540)

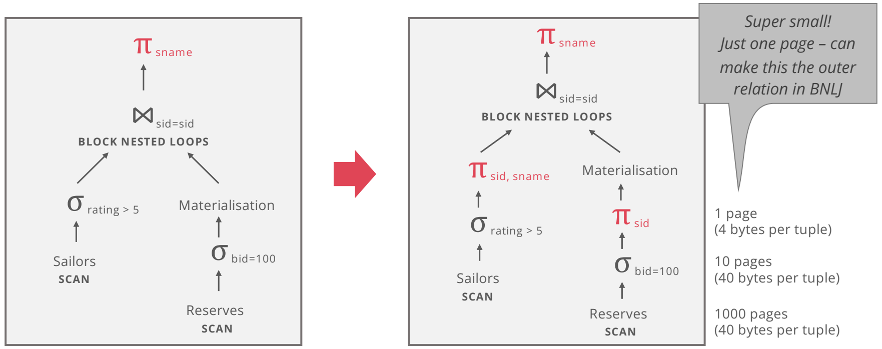
Block nested loops join (2350)

Next: projection cascade





# Projection Cascade & Pushdown




2350 I/Os



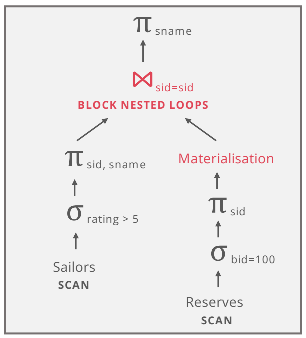


# With Join Reordering, no Mat.

Will try reordering the join again

Will also skip on the materialisation for this

Convince yourself that it doesn't help








# Query Plan 9 Cost

Cost estimation with 5 buffers:

Scan Reserves: **1000 I/Os**

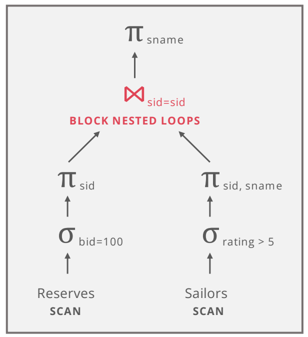
For each block of sids that rented boat 100

Iterate over Sailors: ??? · **500 I/Os**

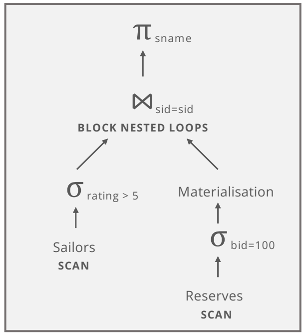
Recall: Reserves tuple is 40B, assume sid is 4B

10 pages down to 1 page

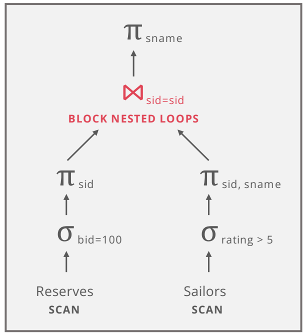
Total = 1000 + 1 · 500 = **1500 I/Os**



# Decision 8




2350 I/Os




1500 I/Os

Cannot do 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)

Materialising inner loop (4250)

Join ordering again with materialisation (4010)

Sort merge join (3540)

Block nested loops join (2350)

Projection cascade, plus reordering again (1500)

Next: indexes



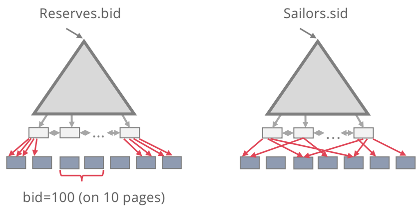


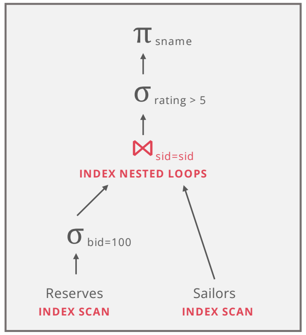
# How About Indexes?

Indexes

- Clustered tree index on Reserves.bid
- Unclustered tree index on Sailors.sid

Assume indexes fit in memory








# How About Indexes?

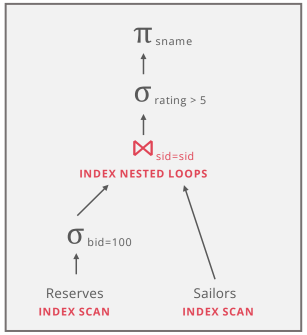
Notes about our query plan:

**No projection pushdown to left for $\pi_{sid}$**

Projecting out unnecessary fields from outer relation of INLJ does not make an I/O difference (still doing things per tuple)

**No selection pushdown to right for $\sigma_{rating > 5}$**

Does not affect Sailors.sid index lookup (I/O cost remains the same)



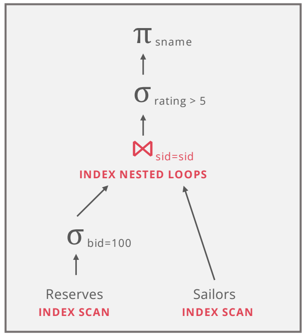
# How About Indexes?

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








# How About Indexes?

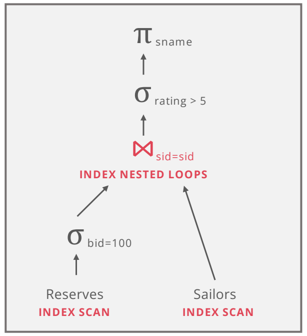
With clustered index on bid of Reserves,
we access how many pages of Reserves?

100,000/100=**1000** tuples on 1000/100=**10** pages

Foreach such Reserves tuple (1000 tuples)

Get matching Sailors tuple (1 I/O)

Total = 10 + 1000 · 1 = 1010 I/Os








# 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)
Materialising inner loop (4250)
Join ordering again with materialisation (4010)
Sort merge join (3540)
Block nested loops join (2350)
Projection cascade, plus reordering again (1500)
Index Nested Loops Join (1010)

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