COMP 3311
DATABASE MANAGEMENT SYSTEMS

LECTURE 16 EXERCISES
QUERY PROCESSING:
JOIN OPERATION

EXERCISE 1

Use the following information about the relations to estimate the page I/O cost to compute the query result using the stated join strategies.

Sailor(sailorId, sName, rating, age)   Reserves(sailorId, boatId, rDate)

Query: Find the names of sailors who have reservations.

```
select sName
from Sailor, Reserves
where Sailor.sailorId=Reserves.sailorId;
```

- Page size: 1000 bytes; buffer size M = 100 pages.
- Each attribute (and pointer where applicable) is 20 bytes.
- Each Sailor tuple is 80 bytes and each Reserves tuple is 60 bytes.
- Sailor tuples = 10,000; $bf_{Sailor} = \lfloor 1000 / 80 \rfloor = 12$ Sailor tuples per page.
- Reserves tuples = 40,000; $bf_{Reserves} = \lfloor 1000 / 60 \rfloor = 16$ Reserves tuples per page.
- Sailor requires $\lceil 10000 / 12 \rceil = 834$ pages and Reserves $\lceil 40000 / 16 \rceil = 2500$ pages.
- Since there are 10,000 Sailor tuples and 40,000 Reserves tuples, a sailor has, on average, 4 reservations (but it is possible that some sailor's have more, while some have none).

EXERCISE 1 (CONT'D)

Sailor tuples: 10,000
bf_{Sailor}: 12
Sailor pages: 834
Reserves tuples: 40,000
$bf_{Reserves}$: 16
Reserves pages: 2500
M pages: 100

a) i. **block nested-loop join** - Sailor as outer relation

- Read 98 pages of the Sailor relation into memory at a time (there are $\lceil 834 / 98 \rceil = \underline{9}$ "blocks" of Sailor pages). (*Requires 98 memory pages*)
- For each "block" of Sailor pages we scan the Reserves relation (page-by-page) to find matching tuples. (*Requires 1 memory page*)
- One memory page is allocated for the output buffer.

Total cost: 9 * 2500 + 834 = 23,334 page I/Os

ii. **block nested-loop join** - Reserves as outer relation

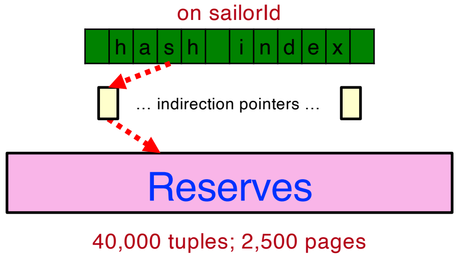
- Read 98 pages of the Reserves relation into memory at a time (there are $\lceil 2500 / 98 \rceil = \underline{26}$ "blocks" of Reserves pages). (*Requires 98 memory pages*)
- For each "block" of Reserves pages we scan the Sailor relation (page-by-page) to find matching tuples. (*Requires 1 memory page*)
- One memory page is allocated for the output buffer.

Total cost: 26 * 834 + 2500 = 24,184 page I/Os

EXERCISE 1 (CONT'D)

Sailor tuples: 10,000
bf_{Sailor}: 12
Sailor pages: 834
Reserves tuples: 40,000
$bf_{Reserves}$: 16
Reserves pages: 2500
M pages: 100

b) **indexed nested-loop join with hash index** on sailorId for the Reserves relation (assume no overflow).



- For each Sailor tuple, find the corresponding entry in the hash index on Reserves.sailorId.
- This takes 1 page I/O per Sailor tuple (since we assume no overflow).
- Since each Sailor tuple has on average 4 reservations, and, since the hash index is non-clustering (secondary), we expect each sailorId to have 4 matching tuples in the Reserves relation. Therefore, we need 4 page I/Os per Sailor tuple to retrieve the Reserves records.
- We also need 1 page I/O per sailorId to retrieve the indirection pointers.

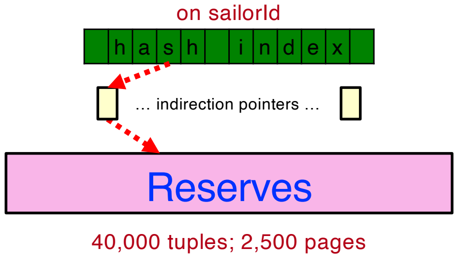
Total cost: cost of reading Sailor + #tuples in Sailor * 6
= 834 + 10,000 * 6 = 60,834 page I/Os

BAD SOLUTION!

EXERCISE 1 (CONT'D)

Sailor tuples: 10,000
bf_{Sailor}: 12
Sailor pages: 834
Reserves tuples: 40,000
$bf_{Reserves}$: 16
Reserves pages: 2500
M pages: 100

GOOD SOLUTION



- Since we do not care about the boats that a Sailor has reserved we can do an index-only scan.
- We do not need to retrieve any tuples from Reserves.
- If a sailor id exists in the index, it means that a sailor has at least one reservation.

Total cost: 834 + 10,000 * 1 = 10,834 page I/Os

Questions:

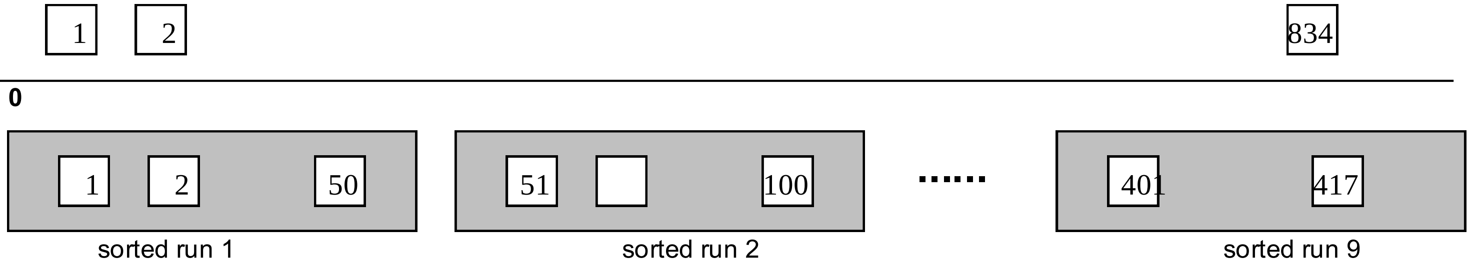
1. What if the above query requested also the boat ids?
2. What if there were an additional condition on Sailor (e.g., **where** Sailor.rating='8').

EXERCISE 1 (CONT'D)

c) merge join

Sort Sailor **on** sailorId

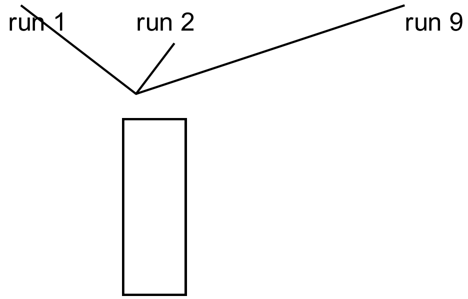
PASS 0



Since there are 100 pages of memory, at each sorted run read in 100 Sailor pages, but only write 50 pages because we discard attributes rating and age since they are not needed for the join and are not required in the result. Totally we write 834/2=417 sorted Sailor pages creating 9 sorted runs.

PASS 1

The Sailor relation can be merged in 1 pass since there are 9 sorted runs, which require only 9 pages.

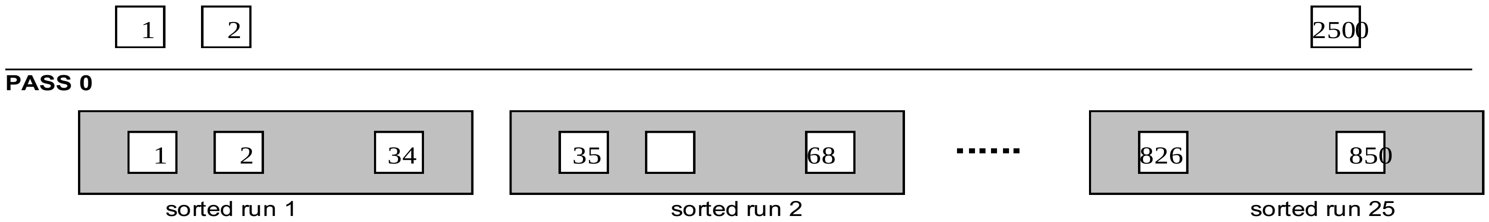


Sailor **sorting cost:** $\underset{r}{834} + \underset{w}{417}$ (Pass 0) + $\underset{r}{417} + \underset{w}{417}$ (Pass 1) = 2,085 page I/Os

EXERCISE 1 (CONT'D)

Sailor tuples: 10,000
bf_{Sailor}: 12
Sailor pages: 834
Reserves tuples: 40,000
$bf_{Reserves}$: 16
Reserves pages: 2500
M pages: 100

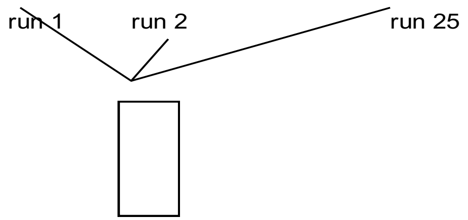
Sort Reserves **on** sailorId



Since there are 100 pages of memory, at each sorted run read in 100 Reserves pages, but only write 34 pages because we discard attributes boatId and date since they are not needed for the join and are not required in the result. Totally we write 34*25=850 sorted Reserves pages creating 25 sorted runs.

PASS 1

The Reserves relation can be merged in 1 pass since there are 25 sorted runs, which require only 25 pages.



At Pass 1 (merge) of the sort, as each sorted page of Reserves is generated we can read the Sailor pages and directly find the joining tuples of Sailor. Consequently, we avoid writing the result of Pass 1 of the sort (i.e., 850 pages) to a temporary file and reading it again for the merge-join phase.

Reserves **sorting cost:** $\underset{r}{2500}$ + $\underset{w}{850}$ (Pass 0) + $\underset{r}{850}$ (Pass 1) = <u>4,200</u> page I/Os

Thus, for joining we only need to read and scan the 417 sorted Sailor pages.

<u>Total cost</u>: 2,085 + 4,200 + 417 = <u>6,702</u> page I/Os

Sailor tuples: 10,000
bf_{Sailor}: 12
Sailor pages: 834
Reserves tuples: 40,000
$bf_{Reserves}$: 16
Reserves pages: 2500
M pages: 100

Optimization!

EXERCISE 1 (CONT'D)

- Considering the relatively large buffer that we have, we could further improve the performance by sorting the two files concurrently. That is, we perform Pass 0 of Sailor (and create 9 sorted runs). Then Pass 0 of Reserves (and create 25 sorted runs).
- Then we sort the two files at the same time (i.e., we allocate 9 buffer pages for the sorted runs of Sailor and 25 buffer pages for Reserves). Once the first sorted page is created during Pass 1 of Sailor, the algorithm will proceed by sorting Reserves and creating the first sorted page. The two pages can be immediately matched without the need to materialize any intermediate file.
- When one of the two pages is exhausted (all its tuples have been matched) the algorithm proceeds by generating the second sorted page for this file and so on.
- This method improves the previous one by avoiding the cost of writing and reading the sorted Sailor result in a temporary file.

Total cost: 6,702 – (2*417) = 5,868 page I/Os

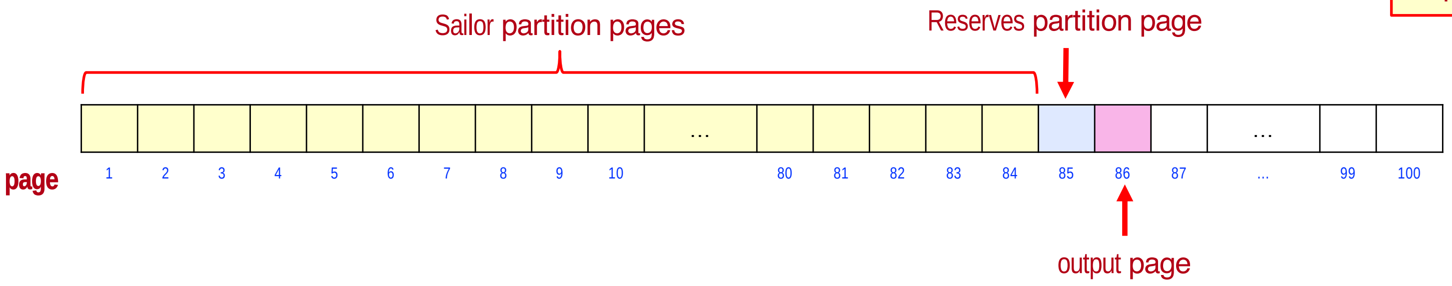
EXERCISE I (CONT'D)

d) **hash join** (assume no overflow)

- Use the smaller relation (Sailor = 834 pages) as the build input.
- We should choose the number of partitions *n* such that, when doing the join phase, all the pages of each build partition of Sailor fits in memory. For example, we can use 10 partitions so that the size of each partition is $\lceil 834/10 \rceil$ = 84 pages, which fit into memory.
- First read and partition Sailor using 10 pages of memory, one for each partition.
  Cost: 834 + 834 = 1,668 page I/Os.
- Then read and partition Reserves using 10 pages of memory, one for each partition.
  Cost: 2500 + 2500 = 5,000 page I/Os.
  - Note that each partition of Reserves occupies 250 pages, which is more than the available memory of 100 pages. However, we don't care since only the partitions of Sailor need to fit in memory.

EXERCISE I (CONT'D)

Sailor tuples: 10,000
bf_{Sailor}: 12
Sailor pages: 834
Reserves tuples: 40,000
$bf_{Reserves}$: 16
Reserves pages: 2500
M pages: 100



– Finally, we read, in turn, each partition (i.e., 84 pages) of Sailor, read the corresponding partition of Reserves page by page, match it (i.e., by hashing) against the Sailor partition and output the matching tuples.
Cost: 834 + 2500 = 3,334 page I/Os.

Total cost: 3 * (2500 + 834) = 10,002 page I/Os

EXERCISE 2

The relations R_1(A, B, C) and R_2(C, D, E) have the following properties:

- R_1 has 20,000 tuples
- R_2 has 45,000 tuples
- 25 tuples of R_1 fit on one page
- 30 tuples of R_2 fit on one page

R_1 requires 800 pages
R_2 requires 1500 pages

Assuming that there are 800 pages of memory available for processing a join, estimate the page I/O cost for each of the following join strategies for R_1 JOIN R_2.

a) nested-loop join
b) block nested-loop join
c) merge join (assume that the relations are not sorted initially)
d) hash join (assume no overflow occurs)

EXERCISE 2 (CONT'D)

R_1 tuples = 20,000
R_1 tuples/page = 25
R_1 pages = 800
R_2 tuples = 45,000
R_2 tuples/page = 30
R_2 pages = 1500
M = 800 pages

a) **nested-loop join:** $\boxed{n_r * B_s + B_r}$

i. using R_1 as the outer relation

Cost: 20000 * 1500 + 800 = 30,000,800 page I/Os

ii. using R_2 as the outer relation

Cost: 45000 * 800 + 1500 = 36,001,500 page I/Os

b) **block nested-loop join:** $\boxed{\lceil B_r / (M\text{-}2) \rceil * B_s + B_r}$

i. using R_1 as the outer relation

Cost: $\lceil 800/798 \rceil$ * 1500 + 800 = 3,800 page I/Os

ii. using R_2 as the outer relation

Cost: $\lceil 1500/798 \rceil$ * 800 + 1500 = 3,100 page I/Os

R_1 tuples = 20,000
R_1 tuples/page = 25
R_1 pages = 800
R_2 tuples = 45,000
R_2 tuples/page = 30
R_2 pages = 1500
M = 800 pages

EXERCISE 2 (CONT'D)

c) **merge join**

Assume R_1 and R_2 are not sorted initially on the join attribute.

Need to use external sorting.

☞ External sorting cost (sort & merge): $\boxed{2*B_r*(1+\lceil \log_{M-1}(B_r/M) \rceil)}$

Sorting cost

Sort R_1: $2*800*(1+\lceil \log_{799}(800/800) \rceil) = 2*800*(1+0) = \underline{1,600}$

Sort R_2: $2*1500*(1+\lceil \log_{799}(1500/800) \rceil) = 2*1500*(1+1) = \underline{6,000}$

Total sort cost: 1600 + 6000 = <u>7,600</u> page I/Os

Merge cost (join phase)

Total join cost: 1500 + 800 = <u>2,300</u> page I/Os

<u>Total cost</u>: 7600 + 2300 = <u>9,900</u> page I/Os

EXERCISE 2 (CONT'D)

R_1 tuples = 20,000
R_1 tuples/page = 25
R_1 pages = 800
R_2 tuples = 45,000
R_2 tuples/page = 30
R_2 pages = 1500
M = 800 pages

d) **hash join**

Assume no overflow occurs.

Use R_1 as the build input since it is smaller.

– Note that there is no need for recursive partitioning since the number of partitions is less than the number of pages M of memory (i.e., $M > \sqrt{B_r} = \sqrt{800} = 28.3$

Total cost: 3 * (1500 + 800) = 6,900 page I/Os