COMP 3311
DATABASE MANAGEMENT SYSTEMS

LECTURE 15 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: select *
from Sailor natural join Reserves;

- 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).

```
select *
from Sailor natural join Reserves;
```

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

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

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

Join page I/O cost: 9 * 2500 + 834 = 23,334

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

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

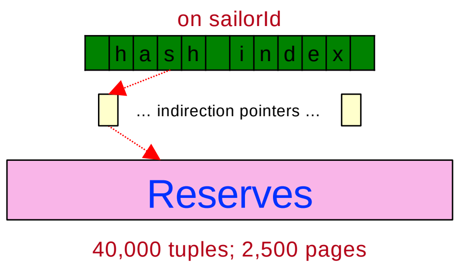
Join page I/O cost: 26 * 834 + 2500 = 24,184

```sql
select *
from Sailor natural join Reserves;
```

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

EXERCISE 1 (cont'd)

b) indexed nested-loop join with hash index on Reserves.sailorId (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.

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

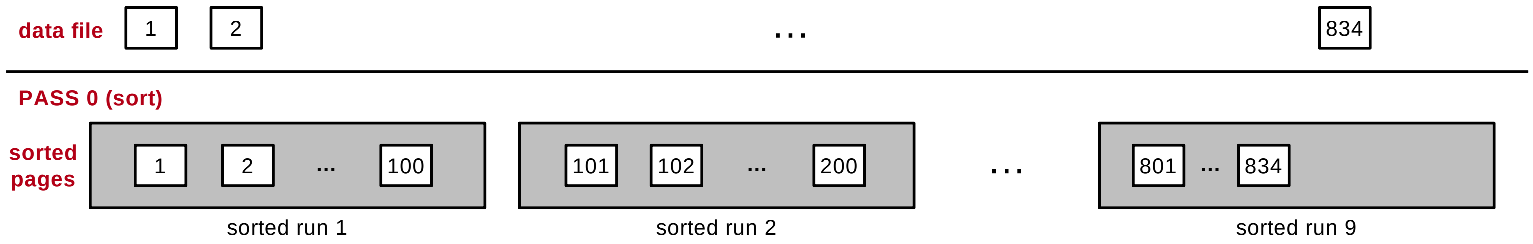
select *
from Sailor natural join Reserves;

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

EXERCISE I (CONT'D)

c) merge join

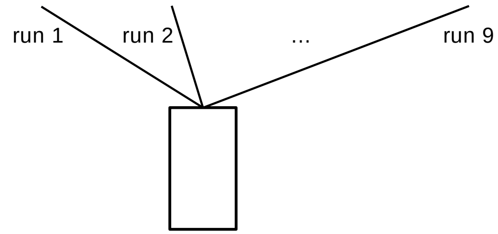
Sort Sailor on sailorId



Since there are 100 buffer pages, for each sorting run read in 100 Sailor pages and write 100 pages. There are $\lceil 834/100 \rceil = 9$ sorted runs created 8 of size 100 pages and 1 of size 34 pages. Totally 834*2=1668 pages are read and written.

PASS 1 (merge)

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



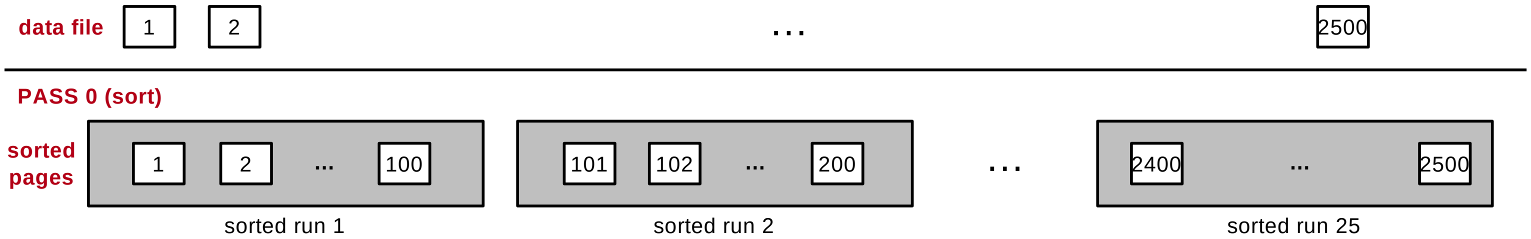
Sailor sorting page I/O cost: $\underset{r}{834} + \underset{w}{834}$ (Pass 0) + $\underset{r}{834} + \underset{w}{834}$ (Pass 1) = 3,336

select *
from Sailor natural join Reserves;

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

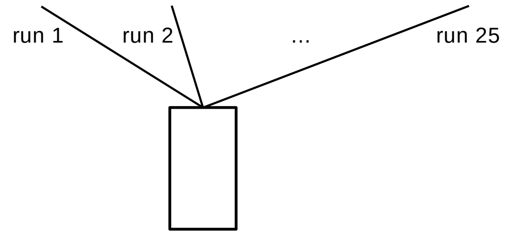
Sort Reserves on sailorId



Since there are 100 buffer pages, for each sorting run read in 100 Reserves pages and write 100 pages. There are $\lceil 2500/100 \rceil$=25 sorted runs created each of size 100 pages. Totally 2500*2=5000 pages are read and written.

PASS 1 (merge)

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., 2500 pages) to a temporary file and reading it again for the merge-join phase.

Reserves sorting page I/O cost: $\underset{r}{2500}$ + $\underset{w}{2500}$ (Pass 0) + $\underset{r}{2500}$ (Pass 1) = 7,500

Thus, for joining we only need to read and scan the 834 sorted Sailor pages.

Join page I/O cost: 3,336 + 7,500 + 834 = 11,670

select *
from Sailor natural join Reserves;

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

EXERCISE I (cont'd)

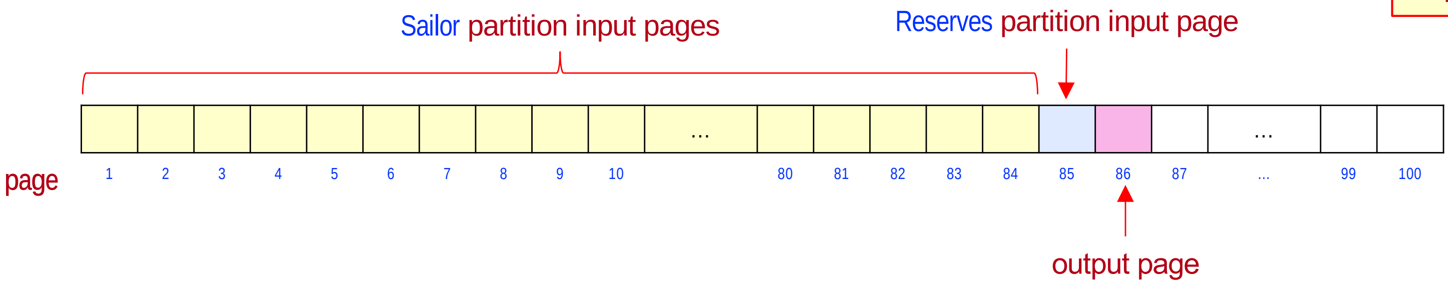
d) hash join (assume no overflow)

- 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 the buffer. For example, we can use 10 partitions so that the size of each partition is $\lceil 834/10 \rceil = 84$ pages, which fit into the buffer.
- First read and partition Sailor using 10 buffer pages, one for each partition.
  Page I/O cost: 834 + 834 = <u>1,668</u> page I/Os.
- Then read and partition Reserves using 10 buffer pages, one for each partition.
  Page I/O cost: 2500 + 2500 = <u>5,000</u> page I/Os.
  - ➢ Note that each partition of Reserves occupies 250 pages, which is more than the available buffer of 100 pages. However, we don't care since only the partitions of Sailor need to fit in the buffer.

select *
from Sailor natural join Reserves;

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

EXERCISE I (cont'd)



– 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.

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

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
- bf_{R1} = 25 tuples
- bf_{R2} = 30 tuples

R_1 requires 800 pages

R_2 requires 1500 pages

Assuming that there are 800 buffer pages 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 <u>not</u> sorted initially)

d) hash join (assume no overflow occurs)

R_1 JOIN R_2

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)

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

i. using R_1 as the outer relation

Join page I/O cost: 20000 * 1500 + 800 = 30,000,800

ii. using R_2 as the outer relation

Join page I/O cost: 45000 * 800 + 1500 = 36,001,500

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

i. using R_1 as the outer relation

Join page I/O cost: $\lceil 800/798 \rceil$ * 1500 + 800 = 3,800

ii. using R_2 as the outer relation

Join page I/O cost: $\lceil 1500/798 \rceil$ * 800 + 1500 = 3,100

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 that both relations are not sorted initially)

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

Page I/O cost to sort R_1: $2*800*(1+\lceil \log_{799}(800/800) \rceil) = 2*800*(1+0) =$ 1,600

Page I/O cost to sort R_2: $2*1500*(1+\lceil \log_{799}(1500/800) \rceil) =$ $2*1500*(1+1) =$ 6,000

Total page I/O cost to sort: 1600 + 6000 = 7,600

Merge cost (join phase)

Total page I/O cost to merge: 1500 + 800 = 2,300

Join page I/O cost: 7600 + 2300 = 9,900

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)

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 buffer pages M (i.e., $M > \sqrt{B_r} = \sqrt{800} = 28.3$

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