COMP 3311
DATABASE MANAGEMENT SYSTEMS

LECTURE 17 EXERCISES
QUERY PROCESSING:
EXPRESSION EVALUATION

EXERCISE 1

The Student relation consists of 10,000 tuples sorted on student id.

Each student has 5 attributes, each 20 bytes, so the tuple size is 100 bytes.

The page size is 1,000 bytes so, $bf_{Student} = \lfloor 1000/100 \rfloor = 10$.

Therefore, $B_{\text{Student}} = \lceil 10000/10 \rceil = 1{,}000$ pages.

Assume that the available main memory M is 100 pages and that there are 5,000 different student names.

There is no index.

We want to evaluate the query:

```
select distinct name
from Student;
```

select distinct name
from Student;

EXERCISE 1 (CONT'D)

Student tuples: 10,000
$bf_{Student}$: 10 tuples/page
$B_{Student}$: 1,000 pages
Each attribute: 20 bytes
Page size: 1,000 bytes
M pages: 100

a) **Projection using external sorting**

Pass 0: Read 100 pages containing 1000 tuples for each sorted run, but write back $\lceil 1000 \text{ tuples} / \lfloor 1000/20 \text{ tuples/page} \rfloor \rceil = \underline{20}$ pages since we only keep name (as other attributes are not needed).

Thus, we create 10 sorted runs of 20 pages each.

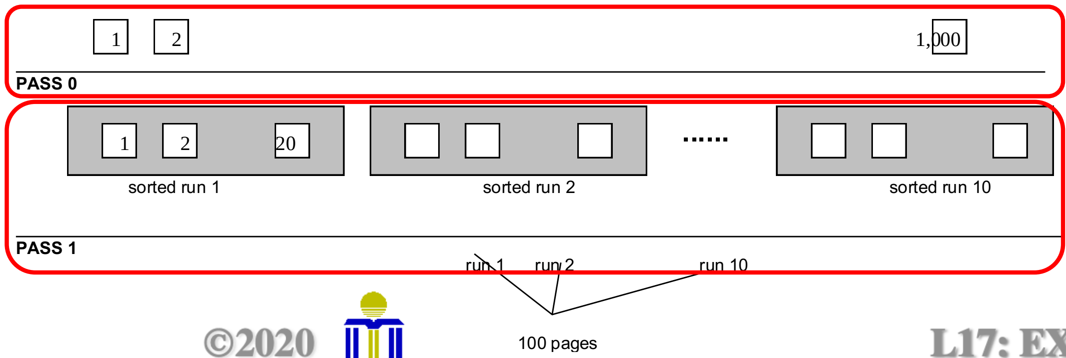
Sort cost: 1000 pages read + 10 * 20 pages written = 1200 page I/Os

Pass 1: Read and merge 10 runs (200 pages) using a 10-way merge.

Merge cost: 200 page I/Os

Total cost: 1200 + 200 = 1400 page I/Os

Result pages: The final output is 200*0.5 = 100 pages since each name is assumed to appear twice (i.e., 5,000 different student names).



```
select distinct name
from Student;
```

Student tuples: 10,000
$bf_{Student}$: 10 tuples/page
$B_{Student}$: 1,000 pages
Each attribute: 20 bytes
Page size: 1,000 bytes
M pages: 100

EXERCISE 1 (CONT'D)

b) **Projection using hashing**

Assume we use 20 buckets (partitions).

Read the file page-by-page and assign each tuple to a bucket.

For each tuple, keep only the name attribute. Thus, read 1,000 pages, but only write back $\lceil 10000 \text{ tuples} / \lfloor 1000/20 \text{ tuples/page} \rfloor \rceil = 200$ pages.

Partitioning cost: 1000 pages read + 200 pages written = 1200 page I/Os

At the next step, we load each bucket into memory, build the in-memory hash table and perform duplicate elimination within the in-memory hash table bucket.

Duplicate elimination cost: 200 page I/Os (to read the 20 buckets)

Total cost: 1200 + 200 = 1400 page I/Os

Result pages: The final output is 200*0.5 = 100 pages since each name is assumed to appear twice (i.e., 5,000 different student names).

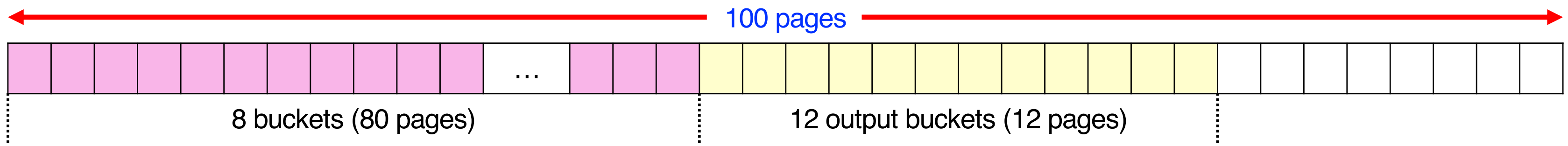
select distinct name
from Student;

EXERCISE 1 (CONT'D)

Student tuples: 10,000
$bf_{Student}$: 10 tuples/page
$B_{Student}$: 1,000 pages
Each attribute: 20 bytes
Page size: 1,000 bytes
M pages: 100

Optimization

Since there are 200 pages written and 20 buckets, the average bucket size is 200 / 20 = 10 pages.



Given the large memory, we can keep 8 full buckets in memory (i.e., 8 x 10 = 80 pages). This leaves 20 pages for the output buffers of which 1 is assigned to each of the remaining 12 buckets that are not kept in memory (i.e., 12 memory pages are assigned as output buffers).

This avoids writing and reading again the 8 buckets (i.e., 80 pages).

Total cost: 1400 - 80 * 2 = 1240 page I/Os
(Compared to 1400 page I/Os.)

EXERCISE 2

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

For the Sailor relation, each tuple is 50 bytes, a page can hold 80 tuples and there are 500 full pages. For the Reserves relation, each tuple is 40 bytes, a page can hold 100 tuples and there are 1,000 full pages. There are 10 different sailor ratings and 100 different boats. Assume that sailors are distributed uniformly over the 10 ratings and reservations are distributed uniformly over the 100 boats.

Our goal is to process the query:

```
select sName
from Sailor natural join Reserves
where boatId=30
      and rating>5;
```

Sailor tuples: 500*80 = 40,000
Reserves tuples: 1,000*100 = 100,000

Some useful statistics

- On average, each sailor has (100,000/40,000) = 2.5 reservations.
- On average, each boat has (100,000/100) = 1,000 reservations.
- On average, for each rating there are (40,000/10) = 4,000 sailors.

EXERCISE 2 (CONT'D)

Estimate the approximate cost of processing the query using a fully pipelined execution method (i.e., do not materialize anything except the final result).

Assume that the Sailor relation contains a clustering B$^+$-tree index with 3 levels on sailorId and the Reserves relation contains a non-clustering hash index on boatId. Both the B$^+$-tree and hash index can fit 400 index entries per page. For non-clustering indexes, each pointer is assumed to lead to a different page. Use Reserves as the outer relation in the join and take advantage of both indexes.

```
select sName
from Sailor natural join Reserves
where boatId=30
      and rating>5;
```

EXERCISE 2 (CONTD)

Size Sailor tuple: 50 bytes
Sailor tuples: 40,000
bf_{Sailor}: 80 tuples/page
B_{Sailor}: 500 pages
Size Reserves tuple: 40 bytes
Reserves tuples: 100,000
$bf_{Reserves}$: 100 tuples/page
$B_{Reserves}$: 1000 pages
Index entries/page: 400

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

There are 10 different sailor ratings and 100 different boats.


```sql
select sName
from Sailor natural join Reserves
where boatId=30
      and rating>5;
```

Some useful statistics

- On average, each sailor has 2.5 reservations.
- On average, each boat has 1,000 reservations.
- On average, for each rating there are 4,000 sailors.

select sName
from Sailor natural join Reserves
where boatId=30
and rating>5;

EXERCISE 2 (CONT'D)

Size Sailor tuple: 50 bytes
Sailor tuples: 40,000
bf_{Sailor}: 80 tuples/page
B_{Sailor}: 500 pages
Size Reserves tuple: 40 bytes
Reserves tuples: 100,000
$bf_{Reserves}$: 100 tuples/page
$B_{Reserves}$: 1000 pages
Index entries/page: 400

a) **Cost to evaluate** $\sigma_{boatId=30}$ (using boatId index on Reserves)

The boat id 30 is hashed requiring 1 page I/O and $\lceil 100{,}000 / 100 \rceil$ = 1000 index entries are returned (since the 100,000 reservations are uniformly distributed over the 100 boats).

The 1000 index entries occupy $\lceil 1000 / 400 \rceil$ = 3 pages requiring 3 page I/Os.

For each index entry, the corresponding Reserves tuple is retrieved requiring 1000 page I/Os (one for each index entry since the index is non-clustering).

We only need to keep the sailor id for each tuple.

~~Cost: 1 + 3 + 1000 = 1004 page I/Os~~

Scanning all Reserves pages requires only 1,000 page I/Os!

b) **Cost to evaluate** $\sigma_{rating>5}$

Cost: 0 page I/Os (see join cost)

```
select sName
from Sailor natural join Reserves
where boatId=30
      and rating>5;
```

EXERCISE 2 (CONT'D)

Size Sailor tuple: 50 bytes
Sailor tuples: 40,000
bf_{Sailor}: 80 tuples/page
B_{Sailor}: 500 pages
Size Reserves tuple: 40 bytes
Reserves tuples: 100,000
$bf_{Reserves}$: 100 tuples/page
$B_{Reserves}$: 1000 pages
Index entries/page: 400

c) **Cost to evaluate** Sailor $\bowtie$ Reserves (indexed nested-loop using sailorId index on Sailor)

For each of the 1000 sailor ids in the result of the selection $\sigma_{boatId=30}$, we use the B+-tree index on sailorId to find the corresponding Sailor tuple.

☞ For each tuple, we check *on-the-fly* if it meets the condition rating>5.

Cost: 1000 * (3 levels + 1 for record retrieval) = 4000 page I/Os

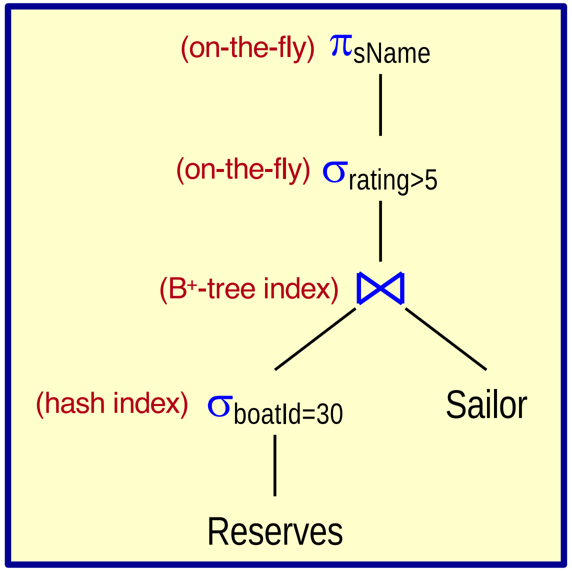
d) **Cost to evaluate** π_{sName}

The projection is done in memory *on-the-fly* as the join is evaluated. Hence, it has no I/O cost.

Cost: 0 page I/Os

Total cost: 1000 + 0 + 4000 + 0
= 5000 page I/Os

Execution plan



EXERCISE 3

The Sailor relation consists of 40,000 tuples sorted on sailor id.

Each sailor has 4 attributes, each 10 bytes, so the tuple size is 40 bytes.

The page size is 800 bytes so $bf_{\text{Sailor}} = \lfloor 800 / 40 \rfloor = 20$.

Therefore, $B_{\text{Sailor}} = \lceil 40000 / 20 \rceil = 2000$ pages.

Assume that the available main memory M is 100 pages and that 5% of sailor names are the same.

There is no index.

We want to evaluate the query:

```
select distinct sName
from Sailor;
```

select distinct sName
from Sailor;

Sailor tuples: 40,000
bf_{Sailor}: 20 tuples/page
B_{Sailor}: 2,000 pages
Page size: 800 bytes
M pages: 100

EXERCISE 3 (CONT'D)

a) **Projection using external sorting**

Pass 0: Read 100 pages containing (100 * 20) = 2000 tuples for each sorted run, but write back $\lceil 2000 / \lfloor 800 / 10 \rfloor \rceil$ = 25 pages since we only keep the name (the other attributes are not needed).

Thus, there are 20 sorted runs of 25 pages each.

Sort cost: 2000 pages read + 20 * 25 pages written = 2500 page I/Os

Pass 1: Read and merge the 500 pages using a 20-way merge.

Merge cost: 500 page I/Os

Total cost: 2500 + 500 = 3000 page I/Os

Result pages: The final output is 500*.95 = 475 pages since it is assumed that 5% of sailor names are the same.

```
select distinct sName
from Sailor;
```

Sailor tuples: 40,000
bf_{Sailor}: 20 tuples/page
B_{Sailor}: 2,000 pages
Page size: 800 bytes
M pages: 100

EXERCISE 3 (CONT'D)

b) **Projection using hashing**

Assume we use 40 buckets.

Read the file page-by-page and assign each tuple to a bucket.

For each tuple, keep only the name attribute (i.e., read 2,000 pages, but only write back $\lceil 40{,}000 / \lfloor 800 / 10 \rfloor \rceil = 500$ pages).

Partitioning cost: 2000 pages read + 500 pages written = 2500 page I/Os

At the next step, load each bucket into memory, build the in-memory hash table and perform duplicate elimination within the in-memory hash table bucket.

Duplicate elimination cost: 500 page I/Os (to read the 40 buckets)

Total cost: 2500 + 500 = 3000 page I/Os

Result pages: The final output is 500*.95 = 475 pages since it is assumed that 5% of sailor names are the same.

EXERCISE 4

Student(sId, name, deptId, address) EnrollsIn(courseId, *sId*, semester, grade)

The Student relation contains 10,000 tuples in 1,000 pages and the EnrollsIn relation contains 50,000 tuples in 5,000 pages. There are 25 different departments and 1,000 different courses. All attributes have the same length. Each available index is a B^+-tree with 3 levels. For non-clustering indexes, each pointer is assumed to lead to a different page.

Our goal is to process the query:

```
select name
from Student natural join EnrollsIn
where courseId='COMP3311'
          and deptId='COMP';
```

Some useful statistics

- On average, a student enrolls in (50,000/10,000) = 5 courses.
- On average, a department has (10,000/25) = 400 students.
- On average, each course has an enrollment of (50,000/1,000) = 50 students.

EXERCISE 4 (CONT'D)

Estimate the approximate cost of processing the query using a fully pipelined execution method (i.e., do not materialize anything except the final result).

Assume that the Student relation contains a clustering index on deptId and the EnrollsIn relation contains a non-clustering index on sId. Use Student as the outer relation in the join and take advantage of both indexes.

```
select name
from Student natural join EnrollsIn
where courseId='COMP3311'
        and deptId='COMP';
```

EXERCISE 4 (CONT'D)

Student tuples: 10,000
$bf_{Student}$: 10 tuples/page
$B_{Student}$: 1000 pages
EnrollsIn tuples: 50,000
$bf_{EnrollsIn}$: 10 tuples/page
$B_{EnrollsIn}$: 5000 pages
EnrollsIn tuples/student: 5

Student(sId, name, deptId, address)

EnrollsIn(courseId, *sId*, semester, grade)

There are 25 different departments and 1000 different courses.


```
select name
from Student natural join EnrollsIn
where courseId='COMP3311'
          and deptId='COMP';
```

Some useful statistics

- On average, a student enrolls in 5 courses.
- On average, a department has 400 students.
- On average, each course has an enrollment of 50 students.

```
select name
from Student natural join EnrollsIn
where courseId='COMP3311'
      and deptId='COMP';
```

EXERCISE 4 (CONT'D)

Student tuples: 10,000
$bf_{Student}$: 10 tuples/page
$B_{Student}$: 1000 pages
EnrollsIn tuples: 50,000
$bf_{EnrollsIn}$: 10 tuples/page
$B_{EnrollsIn}$: 5000 pages
EnrollsIn tuples/student: 5

a) **Cost to evaluate** $\sigma_{courseId='COMP3311'}$

Note that there is no index on courseId.

Cost: 0 page I/Os (see join cost)

b) **Cost to evaluate** $\sigma_{deptId='COMP'}$ (using deptId index on Student):

The selection reads 3 index pages using the clustering index on deptId.

Since the file is ordered on deptId, there are 25 different departments and students are distributed uniformly over departments, $\lceil 1000 / 25 \rceil$ = 40 pages of Student tuples are read.

These 40 pages contain (40 pages * 10 tuples/page) = 400 student tuples.

We only need to keep the student ids and the name for each tuple.

Cost: 3 + 40 = 43 page I/Os

select name
from Student natural join EnrollsIn
where courseId='COMP3311'
and deptId='COMP';

EXERCISE 4 (CONT'D)

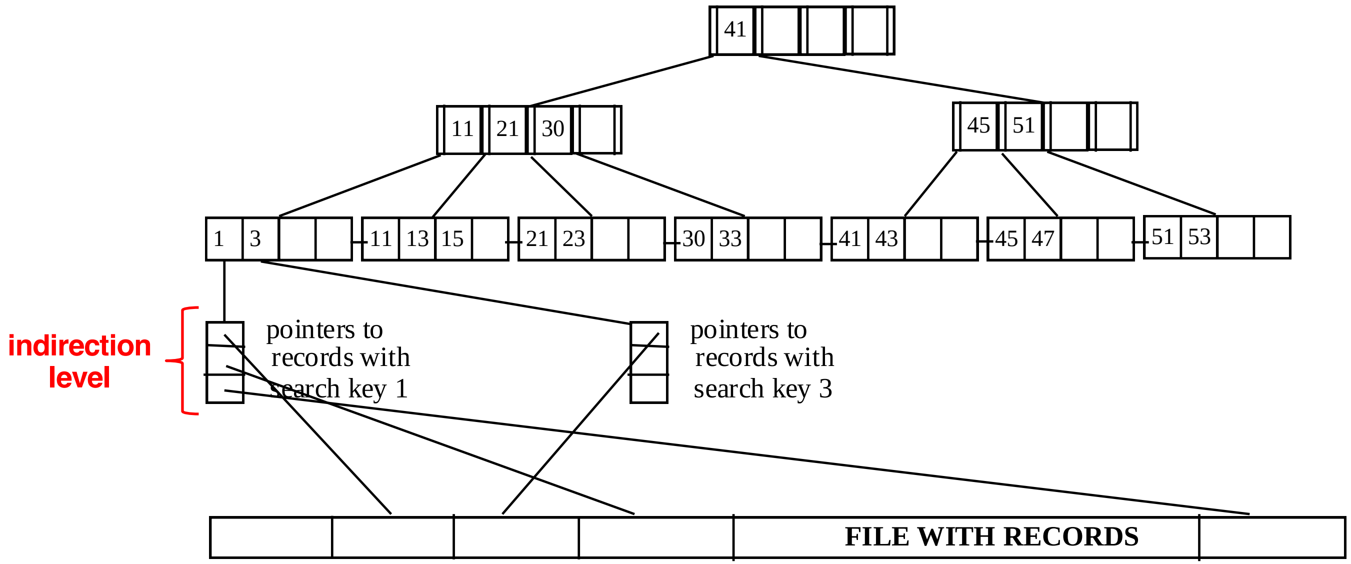
Student tuples: 10,000
$bf_{Student}$: 10 tuples/page
$B_{Student}$: 1000 pages
EnrollsIn tuples: 50,000
$bf_{EnrollsIn}$: 10 tuples/page
$B_{EnrollsIn}$: 5000 pages
EnrollsIn tuples/student: 5

c) **Cost to evaluate** Student ⋈ EnrollsIn (indexed nested-loop using sId index on EnrollsIn)

For each of the 400 students in the selection result, we use the sId index on EnrollsIn to find the corresponding EnrollsIn tuples.

For each sId (student) we need to retrieve 5 EnrollsIn tuples, on average.

Since the sId index on EnrollsIn is non-clustering, we first need to retrieve the pointers to the EnrollsIn records as shown in the figure below.



select name
from Student natural join EnrollsIn
where courseId='COMP3311'
and deptId='COMP';

EXERCISE 4 (CONT'D)

Student tuples: 10,000
$bf_{Student}$: 10 tuples/page
$B_{Student}$: 1000 pages
EnrollsIn tuples: 50,000
$bf_{EnrollsIn}$: 10 tuples/page
$B_{EnrollsIn}$: 5000 pages
EnrollsIn tuples/student: 5

Assuming the 5 record pointers per student all fit on a page (which is highly likely), we need 400 page I/Os to retrieve the record pointers (one for each student).

For each EnrollsIn tuple retrieved, we check *on-the-fly* if it meets the condition courseId='COMP3311'.

Cost: 400*(3+1+5) = 3600 page I/Os

d) **Cost to evaluate π_{name}**

The projection is done in memory *on-the-fly* as the join is evaluated. Hence, it has no I/O cost.

Cost: 0 page I/Os

Total cost: 43 + 0 + 3600 + 0
= 3643 page I/Os

Execution plan