COMP 3311
DATABASE MANAGEMENT SYSTEMS

LECTURE 19 EXERCISES
QUERY OPTIMIZATION

EXERCISE 1

Given relation R(A, B, C)

Assume: R contains 10,000 tuples in 1,000 pages.

A has 50 distinct values in the range 1…50.

B has 100 distinct values in the range 0…100.

Estimate the size, *SC* (number of tuples), of each of the following operations *assuming uniform distribution and attribute independence*.

a) $\sigma_{A=10}R$

b) $\sigma_{A=10 \wedge 20 \leq B}R$

c) $\sigma_{C=1}R$

d) $\sigma_{C=10 \wedge A=10}R$

e) $\sigma_{C=10 \wedge A=10 \wedge 20 \leq B}R$

EXERCISE 1 (CONT'D)

n_r: 10,000
B_r: 1000
tuples/page: 10
V(A, R): 50
V(B, R): 100

R(A, B, C)

a) $\sigma_{A=10}R$

Estimated size: $SC(A{=}10, R) = n_r \,/\, V(A, r) = 10{,}000 \,/\, 50 = \underline{200}$ tuples

b) $\sigma_{A=10 \wedge 20<B}R$

Selectivity(A=10) is $SC(A{=}10, R) \,/\, n_r = (n_r \,/\, V(A, R)) \,/\, n_r = 1 \,/\, V(A, R) = 1 \,/\, 50$

Selectivity(20≤B) is $SC(20 \le B, R) \,/\, n_r = n_r * \left(\frac{\max(A,r) - v}{\max(A,r) - \min(A,r)}\right) \,/\, n_r = 80 \,/\, 100$

Estimated size: $SC(A{=}10 \wedge 20 \le B, R) = n_r * (SC(A{=}10, R) \,/\, n_r * SC(20 \le B, R) \,/\, n_r)$

$= 10{,}000 * (1 \,/\, 50 * 80 \,/\, 100)$

$= 10{,}000 * (\, 8 \,/\, 500 \,) = \underline{160}$ tuples

c) $\sigma_{C=1}R$

Estimated size: $SC(C{=}1, R) = n_r \,/\, V(C, R) = 10{,}000 * 1 \,/\, 10{,}000 = \underline{1}$ tuple

(since C is the primary key)

EXERCISE 1 (CONT'D)

n_r: 10,000
B_r: 1000
tuples/page: 10
$V(A, R)$: 50
$V(B, R)$: 100

R(A, B, <u>C</u>)

d) $\sigma_{C=10 \wedge A=10} R$

Selectivity(C=10) = 1 / 10,000

Selectivity(A=10) = 1 / 50

Overall selectivity = 1 / 10,000 * 1 / 50 = 1 / 500,000

Estimated size: 10,000 * 1 / 500,000 = <u>0.02</u> tuples (or a maximum of 1).

e) $\sigma_{C=10 \wedge A=10 \wedge 20<B} R$

Selectivity(C=10) = 1 / 10,000

Selectivity(A=10) = 1 / 50

Selectivity(20≤B) = 8 / 10

Overall selectivity = 1 / 10,000 * 1 / 50 * 8 / 10 = 8 / 5,000,000

Estimated size: 10,000 * 8 / 5,000,000
= <u>0.016</u> tuples (or a maximum of 1).

EXERCISE 2

Consider the relation Sailor(sailorId, sName, rating, age).

n_{Sailor} = 10,000 tuples

B_{Sailor} = 1,000 pages

$bf_{Sailor} = \lceil 10{,}000 / 1{,}000 \rceil = 10$ tuples/page

$V(\text{rating}, \text{Sailor}) = 10$ (10 distinct rating values)

$V(\text{age}, \text{Sailor}) = 100$ (100 distinct age values)

```
select sName
from Sailor
where rating=7
      and age=40;
```

$SC(\text{rating}=7, \text{Sailor}) = n_{Sailor} / V(\text{rating}=7, \text{Sailor}) = 10{,}000 / 10 = \underline{1{,}000}$ tuples

$SC(\text{age}=40, \text{Sailor}) = n_{Sailor} / V(\text{age}=40, \text{Sailor}) = 10{,}000 / 100 = \underline{100}$ tuples

Estimate the cost of the following alternative plans to process the query assuming uniform distribution and attribute independence. *Ignore the cost of searching any indexes.*

a) file scan

b) binary search

c) single B^+-tree index (on either attribute)

d) multiple B^+-tree indexes (on both rating and age)

```
select sName
from Sailor
where rating=7
      and age=40;
```

EXERCISE 2 (CONT'D)

n_{Sailor}: 10,000 tuples
B_{Sailor}: 1,000 pages
bf_{Sailor}: 10 tuples/page
V(rating, Sailor): 10 distinct values
V(age, Sailor): 100 distinct values
SC(rating=7, Sailor) = 1,000 tuples
SC(age=40, Sailor) = 100 tuples

a) file scan

Read the whole Sailor relation and select the records that satisfy both conditions.

Cost: B_{Sailor} = 1,000 page I/Os

b) binary Search

If the file is sorted on rating (or age), do a binary search and find the first record satisfying the condition rating=7 (or age=40).

Retrieve the remaining records sequentially and filter out non-qualifying tuples.

Question: Which is cheaper, searching on rating or age?

```
select sName
from Sailor
where rating=7
      and age=40;
```

EXERCISE 2 (CONTD)

n_{Sailor}: 10,000 tuples
B_{Sailor}: 1,000 pages
bf_{Sailor}: 10 tuples/page
V(rating, Sailor): 10 distinct values
V(age, Sailor): 100 distinct values
SC(rating=7, Sailor) = 1,000 tuples
SC(age=40, Sailor) = 100 tuples

Cost to search on rating

Cost: $\underbrace{\lceil \log_2(B_{Sailor}) \rceil}_{\text{\# I/Os to find 1st page}} + \underbrace{\lceil SC(\text{rating=7, Sailor}) / bf_{Sailor} \rceil}_{\text{\# I/Os to retrieve all pages}} - \underbrace{1}_{\text{1st page I/O}}$

$= \lceil \log_2(1{,}000) \rceil + \lceil 1{,}000 / 10 \rceil - 1$

$= 10 + 99$

= <u>109</u> page I/Os

Cost to search on age

Cost: $\underbrace{\lceil \log_2(B_{Sailor}) \rceil}_{\text{\# I/Os to find 1st page}} + \underbrace{\lceil SC(\text{age=40, Sailor}) / bf_{Sailor} \rceil}_{\text{\# I/Os to retrieve all pages}} - \underbrace{1}_{\text{1st page I/O}}$

$= \lceil \log_2(1{,}000) \rceil + \lceil 100 / 10 \rceil - 1$

$= 10 + 9$

= <u>19</u> page I/Os

```
select sName
from Sailor
where rating=7
      and age=40;
```

EXERCISE 2 (CONT'D)

n_{Sailor}: 10,000 tuples
B_{Sailor}: 1,000 pages
bf_{Sailor}: 10 tuples/page
V(rating, Sailor): 10 distinct values
V(age, Sailor): 100 distinct values
SC(rating=7, Sailor) = 1,000 tuples
SC(age=40, Sailor) = 100 tuples

c) single B$^+$-tree index (on either attribute)

Use the index for one of the two conditions and check the other condition in memory.

☞ **Not necessarily a good solution if the index is not clustered.**

```
select sName
from Sailor
where rating=7
      and age=40;
```

EXERCISE 2 (CONTD)

n_{Sailor}: 10,000 tuples
B_{Sailor}: 1,000 pages
bf_{Sailor}: 10 tuples/page
V(rating, Sailor): 10 distinct values
V(age, Sailor): 100 distinct values
SC(rating=7, Sailor) = 1,000 tuples
SC(age=40, Sailor) = 100 tuples

Index on rating

Use the index to find records where rating=7.

For each qualifying record, check the condition age=40.

- For rating=7, V(rating, Sailor) = 10 distinct values.
- SC(rating=7, Sailor) = n_{Sailor} / V(rating=7, Sailor)
  = 10,000 / 10 = 1,000 tuples.
- If the index on rating is not clustered, to retrieve these sailors we need 1,000 random page I/Os.

  ☞ **Might as well do a sequential scan!**
- If the index is clustered, all sailors with rating=7 are in $\lceil 1{,}000 / 10 \rceil$ = 100 consecutive pages.

```
select sName
from Sailor
where rating=7
      and age=40;
```

EXERCISE 2 (CONTD)

n_{Sailor}: 10,000 tuples
B_{Sailor}: 1,000 pages
bf_{Sailor}: 10 tuples/page
V(rating, Sailor): 10 distinct values
V(age, Sailor): 100 distinct values
SC(rating=7, Sailor) = 1,000 tuples
SC(age=40, Sailor) = 100 tuples

Index on age

Use the index to find records where age=40.

For each qualifying record, check the condition rating=7.

- For age=40, V(age, Sailor) = 100 distinct values.
- SC(age=40, Sailor) = n_{Sailor} / V(age=40, Sailor)
  = 10,000 / 100 = 100 tuples.
- If the index on rating is not clustered, to retrieve these sailors we need 100 random page I/Os.
- If the index is clustered, all sailors with age=40 are in $\lceil 100 / 10 \rceil$ = 10 consecutive pages.

```
select sName
from Sailor
where rating=7
      and age=40;
```

EXERCISE 2 (CONT'D)

n_{Sailor}: 10,000 tuples
B_{Sailor}: 1,000 pages
bf_{Sailor}: 10 tuples/page
V(rating, Sailor): 10 distinct values
V(age, Sailor): 100 distinct values
SC(rating=7, Sailor) = 1,000 tuples
SC(age=40, Sailor) = 100 tuples

d) multiple B^+-tree indexes (on both rating and age)

Use more than one index and intersect record pointers (rids) before retrieval of actual tuples.

- Use the index on age to find all rids of tuples where age=40.
  - Selectivity(age=40) = SC(age=40, Sailor) / n_{Sailor} = 100 / 10,000 = 0.01.
- Use the index on rating to find all rids of tuples where rating=7.
  - Selectivity(rating=7) = SC(rating=7, Sailor) / n_{Sailor} = 1,000 / 10,000 = 0.1.

Estimated size: n_{Sailor} * selectivity(age=40) * selectivity(rating=7)
= 10,000 * 0.01 * 0.1 = 10 tuples.

Cost: 10 page I/Os (as each tuple could be on a different page)

EXERCISE 3

Employee(empId: 4 bytes, name: 35 bytes, title: 2 bytes, salary: 5 bytes, deptId: 4 bytes)
Employee: 50 bytes/tuple; 20,000 tuples; 250 pages

Department(deptId: 4 bytes, projectId: 4 bytes, name: 25 bytes, location: 7 bytes)
Department: 40 bytes/tuple; 500 tuples; 5 pages

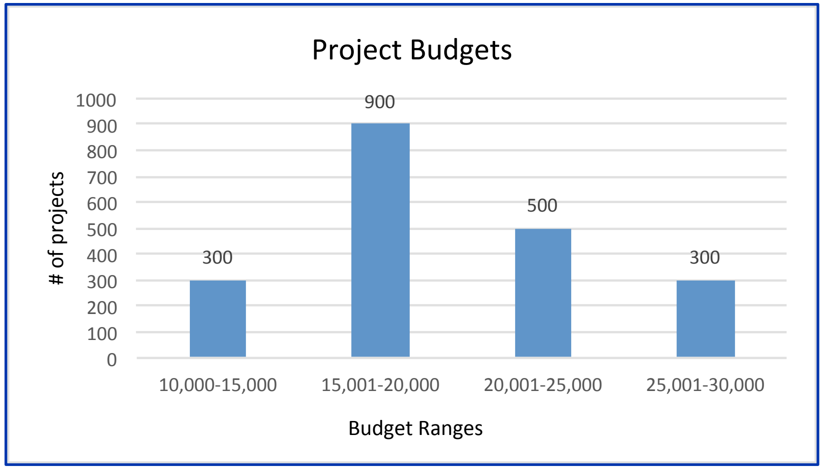
Project(projectId: 4 bytes, title: 20 bytes, budget: 6 bytes, report: 970 bytes)
Project: 1,000 bytes/tuple; 2,000 tuples; 500 pages

Employee salaries: uniformly distributed in the range 10,000 to 110,000.

Project budgets: distributed in the range 10,000 to 30,000 according to the histogram.

Page size: 4,000 bytes Memory buffer pages: 12

- There is a clustering B+-tree index with 3 levels on salary for Employee.
- There is a hash index on deptId for Department, which is ordered on deptId.
- There is a hash index on projectId for Project, which is ordered on projectId.

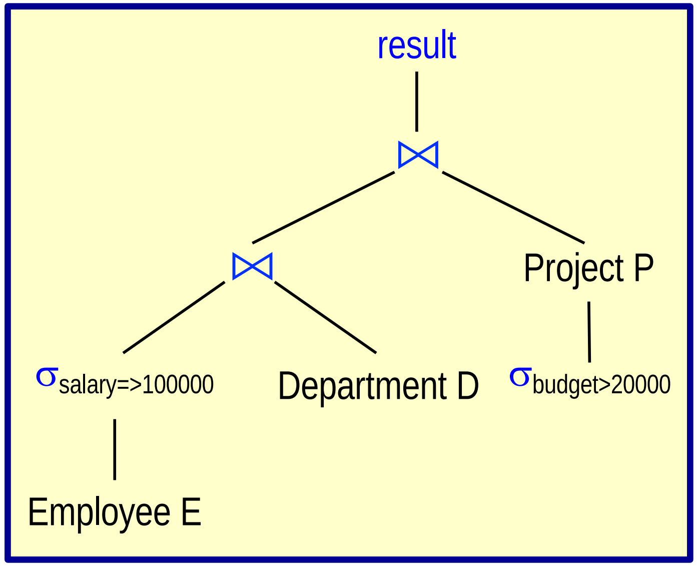


EXERCISE 3 (CONT'D)

a) Use the relational algebra tree to estimate the output size of the query in tuples.

b) Evaluate the query using the relational algebra tree and the steps given below. The goal is to minimize the average number of page I/Os. Where possible, use pipelining rather than materialization (i.e., keep intermediate results in memory where possible). Assume the file organizations and indexes described above. For each step, give the strategy used and the average case page I/O cost. Give the total page I/O cost to process the query.

```
select E.empId, D.deptId, P.projectId
from Employee E, Department D, Project P
where E.deptId=D.DeptId
        and D.projectId=P.projectId
        and salary=>100000
        and budget>20000;
```



```
select E.empId, D.deptId, P.projectId
from Employee E, Department D, Project P
where E.deptId=D.DeptId
      and D.projectId=P.projectId
      and salary=>100000
      and budget>20000;
```

EXERCISE 3 (CONTD)

$n_{Employee}$: 20,000 tuples
$B_{Employee}$: 250 pages
$n_{Department}$: 500 tuples
$B_{Department}$: 5 pages
$n_{Project}$: 1,000 tuples
$B_{Project}$: 500 pages
Page size: 4,000 bytes
M: 12 pages

a) Estimate the output size of the query in tuples.

For $\sigma_{salary=>100000}$Employee $\Longrightarrow$ result A (salaries distributed uniformly)

Selectivity: $\frac{\max(A,r)-v}{\max(A,r)-\min(A,r)} = \frac{110{,}000-100{,}000}{110{,}000-10{,}000} = \frac{1}{10}$

Estimated size: 20,000 * $\frac{1}{10}$ = 2000 tuples

For result A $\bowtie$ Department $\Longrightarrow$ result B (join is on the foreign key of Employee)

Every tuple of result A will join with exactly one tuple of Department.

Estimated size: 2000 tuples

For $\sigma_{budget>20000}$Project $\Longrightarrow$ result C (budgets distributed according to histogram)

Selectivity: 500 + 300 / 2000 = 0.4

For result B $\bowtie$ result C (join is on the foreign key of Department) in result B

Every tuple of result B will join with exactly one tuple of Result C, but only 0.4 of these tuples will meet the condition $\sigma_{budget>20000}$.

Estimated size: 2000 * 0.4 = 800 tuples

Estimated output size: 800 tuples

```
select E.empId, D.deptId, P.projectId
from Employee E, Department D, Project P
where E.deptId=D.DeptId
      and D.projectId=P.projectId
      and salary=>100000
      and budget>20000;
```

EXERCISE 3 (CONT'D)

$n_{Employee}$: 20,000 tuples
$B_{Employee}$: 250 pages
$n_{Department}$: 500 tuples
$B_{Department}$: 5 pages
$n_{Project}$: 1,000 tuples
$B_{Project}$: 500 pages
Page size: 4,000 bytes
M: 12 pages

b) Evaluate the query.

Step 1: $\sigma_{salary=>100000}$Employee $\Longrightarrow$ result A

Strategy: index lookup using B+-tree on salary

Use the B+-tree to find the first page where the salary equals 100,000. The Employee table occupies $\lceil 20000 / \lfloor 4000/50 \rfloor \rceil = 250$ pages. Since the table is ordered on salary and from a) we know that 1/10 of employees have a salary ≥ 100,000, then 250 * 0.1 = 25 pages contain Employee records where the salary is ≥ 100,000.

Therefore, the cost is 3 pages I/Os (to search the B+-tree) plus 25 page I/Os to retrieve all the qualifying Employee records.

Cost: 3 + 25 = 28 page I/Os

From a) we expect to retrieve 2,000 tuples. We need to keep only empId and deptId which will occupy $\lceil 2000 / \lfloor 4000 / 8 \rfloor \rceil = 4$ pages and which are kept in memory to join with Department in the next step.

```
select E.empId, D.deptId, P.projectId
from Employee E, Department D, Project P
where E.deptId=D.DeptId
      and D.projectId=P.projectId
      and salary=>100000
      and budget>20000;
```

EXERCISE 3 (CONT'D)

$n_{Employee}$: 20,000 tuples
$B_{Employee}$: 250 pages
$n_{Department}$: 500 tuples
$B_{Department}$: 5 pages
$n_{Project}$: 1,000 tuples
$B_{Project}$: 500 pages
Page size: 4,000 bytes
M: 12 pages

Step 2: result A $\bowtie$ Department $\Longrightarrow$ result B

Strategy 1: indexed nested-loop join using deptId hash index

For each of the 2,000 tuples from Step 1, use the deptId hash index on Department to join each tuple with the matching Department tuple. The cost to do this is 1 page I/O to the index and 1 page I/O to retrieve the Department record for each of the 2,000 tuples.

Cost: 2000 * 2 = 4000 page I/Os

Strategy 2: block nested-loop join

The Department table occupies $\lceil 500 / \lfloor 4000/40 \rfloor \rceil$ = 5 pages. These 5 pages can be read into memory and checked whether there is a match with a tuple from the previous step using block nested loop join.

Cost: 5 page I/Os

Each of the 2,000 tuples from Step 1 is expected to match with a Department tuple. We need to keep only empId, deptId and projectId which will occupy $\lceil 2000 / \lfloor 4000 / 12 \rfloor \rceil$ = 7 pages and which are kept in memory to join with Project.

```
select E.empId, D.deptId, P.projectId
from Employee E, Department D, Project P
where E.deptId=D.DeptId
      and D.projectId=P.projectId
      and salary=>100000
      and budget>20000;
```

EXERCISE 3 (CONT'D)

$n_{Employee}$: 20,000 tuples
$B_{Employee}$: 250 pages
$n_{Department}$: 500 tuples
$B_{Department}$: 5 pages
$n_{Project}$: 1,000 tuples
$B_{Project}$: 500 pages
Page size: 4,000 bytes
M: 12 pages

Step 3: $\sigma_{budget>20000}$Project $\Longrightarrow$ result C

Strategy: do on-the-fly with join

Cost: 0 page I/Os

Step 4: result B $\bowtie$ result C $\equiv$ result B $\bowtie$ $\sigma_{budget>20000}$Project

Strategy 1: indexed nested-loop join using projectId hash index

For each of the 2,000 tuples from Step 2, use the projectId hash index on Project to join each tuple with the matching Project tuple. The cost to do this is 1 page I/O to the index and 1 page I/O to retrieve the Project record for each of the 2,000 tuples.

Cost: 2000 * 2 = 4000 page I/Os

```
select E.empId, D.deptId, P.projectId
from Employee E, Department D, Project P
where E.deptId=D.DeptId
      and D.projectId=P.projectId
      and salary=>100000
      and budget>20000;
```

EXERCISE 3 (CONT'D)

$n_{Employee}$: 20,000 tuples
$B_{Employee}$: 250 pages
$n_{Department}$: 500 tuples
$B_{Department}$: 5 pages
$n_{Project}$: 1,000 tuples
$B_{Project}$: 500 pages
Page size: 4,000 bytes
M: 12 pages

Strategy 2: block nested-loop join

The Project table occupies $\lceil 2000 / \lfloor 4000/1000 \rfloor \rceil = 500$ pages. These 500 pages can be read into memory one-by-one, checked whether there is a match with the condition and, if there is, joined with the tuples of result B, which are also in memory, using block nested-loop join.

Cost: 500 page I/Os to read Project and join with result B

For both of the previous strategies, each of the 2,000 tuples from Step 2 is expected to match with a Project tuple. However, given the selection condition $\sigma_{budget>20000}$, it is expected that only 40% of the Project tuples that match with the 2,000 tuples will meet the selection condition.

Therefore, 2000 * .4 = 800 tuples will qualify for the final result.

Total cost: 28 + 5 + 500 = 533 page I/Os

For the final result we need to keep only empId, deptId and projectId which will occupy $\lceil 800 / \lfloor 4000 / 12 \rfloor \rceil = 3$ pages.

Output result size: 3 pages