COMP 3311
DATABASE MANAGEMENT SYSTEMS

LECTURE 18 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* (i.e., 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 I (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>)

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

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 I (CONT'D)

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

R(A, B, C)

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 = 0.02 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
= 0.016 tuples (or a maximum of 1).

EXERCISE 2

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

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

n_{Sailor} = 10,000 tuples

B_{Sailor} = 1,000 pages

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

V(rating, Sailor) = 10 (10 distinct rating values)

V(age, Sailor) = 100 (100 distinct age values)

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

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

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

a) file scan – linear search

b) file scan – 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 – linear search

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

Page I/O cost: B_{Sailor} = 1,000

b) file scan – 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 tuples that do not meet the other condition.

Question: Which is cheaper, searching on rating or 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

i. Search on rating

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

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

$= 10 + 99$

$= \underline{109}$

ii. Search on age

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

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

$= 10 + 9$

$= \underline{19}$

```
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 (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

i. 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.
- Sailor has 1,000 pages.

  ☞ **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 (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

ii. 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.
- Sailor has 1,000 pages.

  ☞ **Better than a sequential scan!**
- 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.

Page I/O cost: 10 (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, deptName: 25 bytes, location: 7 bytes)
Department: 40 bytes/tuple; 100 tuples; 1 page

DeptProject(deptId: 4 bytes, projectId: 4 bytes)
DeptProject: 8 bytes/tuple; 4,000 tuples; 8 pages

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

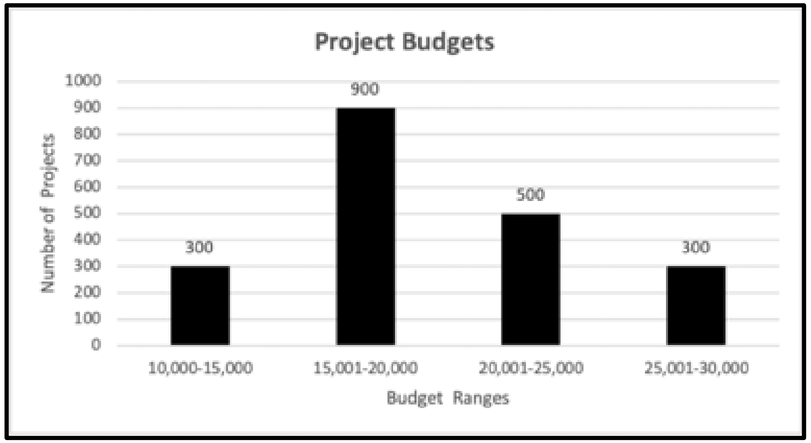
Employee salaries: uniformly distributed in the range 10,000 to 110,000.

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

All foreign keys are not null.

Page size: 4,000 bytes; 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.

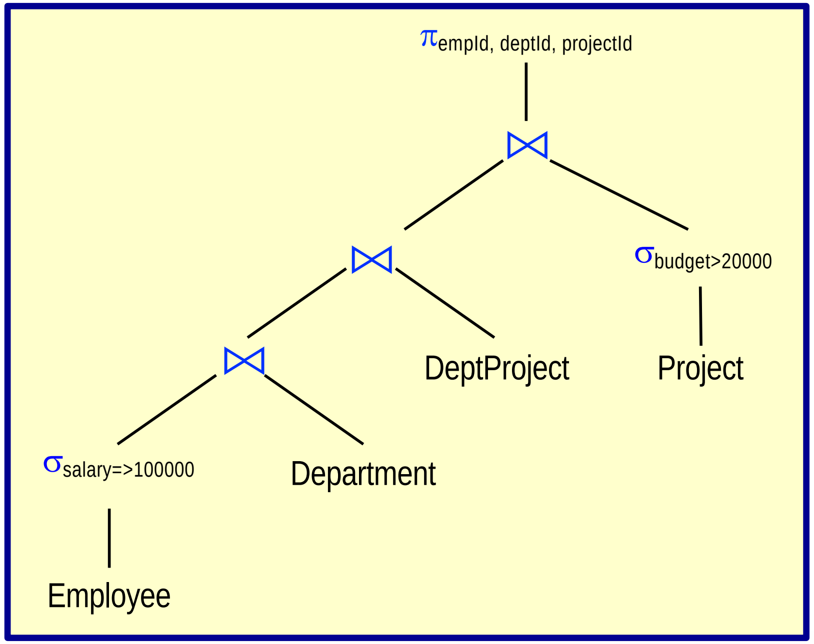


EXERCISE 3 (CONT'D)

a) Estimate the output size of the query in tuples and in pages for the given relational algebra tree.

b) Estimate the query processing page I/O cost using the relational algebra tree and steps given below. The goal is to minimize the average number of page I/Os. Reorder operations, as necessary, to reduce the page I/O cost. 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 query processing page I/O cost.

```
select distinct empId, deptId, projectId
from Employee natural join Department
        natural join DeptProject
        natural join Project
where salary=>100000
         and budget>20000;
```



```
select distinct empId, deptId, projectId
from Employee natural join Department
      natural join DeptProject
      natural join Project
where salary=>100000
       and budget>20000;
```

EXERCISE 3 (cont'd)

Employee(empId: 4 bytes, name, title, salary, deptId)
Department(deptId: 4 bytes, deptName, location)
DeptProject(deptId: 4 bytes, projectId: 4 bytes)
Project(projectId: 4 bytes, projectTitle, budget, report)

$n_{Employee}$: 20,000 tuples
$B_{Employee}$: 250 pages
$n_{Department}$: 100 tuples
$B_{Department}$: 1 pages
$n_{DeptProject}$: 4,000
$B_{DeptProject}$: 8 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 and in pages.

For $\sigma_{salary=>100000}$Employee ⟹ result A (retain only (empId, deptId) 8 bytes)

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 (tuples): $20{,}000 * \frac{1}{10} = \underline{2{,}000}$

Estimated size (pages): $= \lceil 2{,}000 / \lfloor 4000 / 8 \rfloor \rceil = \underline{4}$

For result A ⋈ Department ⟹ result B (join is on foreign key of Employee in Result A)

Since deptId in Employee is a not null foreign key referencing Department, every tuple of result A joins with exactly one tuple of Department. Therefore, the size of result B is the same as that of result A. Retain only (empId, deptId) 8 bytes.

Estimated size (tuples): <u>2,000</u> Estimated size (pages): <u>4</u>

For result B ⋈ DeptProject ⟹ result C (deptId is not a key in result B or DeptProject)

Since every Department tuple is related to 4,000 / 100 = 40 DeptProject tuples on average, every result B tuple will join with 40 DeptProject tuples. Retain only (empid, deptId, projectId) 12 bytes.

Estimated size (tuples): $n_{\text{Result B}} * n_{DeptProject} / V(\text{deptId, Department})$
$= 2{,}000 * 4{,}000 / 100 = \underline{80{,}000}$

Estimated size (pages): $\lceil 80{,}000 / \lfloor 4000 / 12 \rfloor \rceil = 241$

```
select distinct empId, deptId, projectId
from Employee natural join Department
        natural join DeptProject
        natural join Project
where salary=>100000
        and budget>20000;
```

EXERCISE 3 (CONT'D)

Employee(empId: 4 bytes, name, title, salary, deptId)
Department(deptId: 4 bytes, deptName, location)
DeptProject(deptId: 4 bytes, projectId: 4 bytes)
Project(projectId: 4 bytes, projectTitle, budget, report)

$n_{Employee}$: 20,000 tuples
$B_{Employee}$: 250 pages
$n_{Department}$: 100 tuples
$B_{Department}$: 1 pages
$n_{DeptProject}$: 4,000
$B_{DeptProject}$: 8 pages
$n_{Project}$: 1,000 tuples
$B_{Project}$: 500 pages
Page size: 4,000 bytes
M: 12 pages

For $\sigma_{budget>20000}$Project $\Longrightarrow$ result D (budgets distributed as in histogram)

Selectivity: 500 + 300 / 2000 = 0.4

For result C $\bowtie$ result D (join is on foreign key of DeptProject in result C)

Every tuple of result C will join with a tuple of Result D (i.e., with a Project tuple), but only 0.4 of these tuples will meet the condition $\sigma_{budget>20000}$.

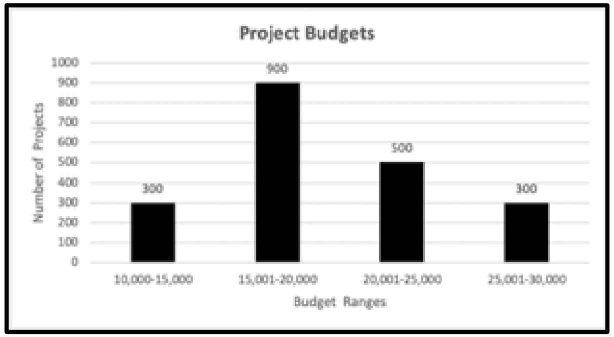
Estimated size: $n_{Result\ C}$ * 0.4 = 80,000 * 0.4 = 32,000

Estimated size (pages): $\lceil 32{,}000 / \lfloor 4000 / 12 \rfloor \rceil$ = 97

Since every employee is related to a unique department and every department is related to a unique set of projects, the combination of (empId, deptId, projectId) in the query result is unique (i.e., there are no duplicates).

Estimated query result output size (tuples): 32,000

Estimated query result output size (pages): 97



select distinct empId, deptId, projectId
from Employee natural join Department
natural join DeptProject
natural join Project
where salary=>100000
and budget>20000;

EXERCISE 3 (CONT'D)

$n_{Employee}$: 20,000 tuples
$B_{Employee}$: 250 pages
$n_{Department}$: 100 tuples
$B_{Department}$: 1 pages
$n_{DeptProject}$: 4,000
$B_{DeptProject}$: 8 pages
$n_{Project}$: 1,000 tuples
$B_{Project}$: 500 pages
Page size: 4,000 bytes
M: 12 pages
B⁺-tree: Employee.salary
Hash index: Department.deptId
Hash index: Project.projectId

b) Estimate the query processing page I/O cost.

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

Strategy 1: file scan – linear search

Strategy 1 page I/O cost: 250

Strategy 2: index lookup using B⁺-tree on salary

Use the B⁺-tree to find the first page where the salary equals 100,000. The Employee relation occupies 250 pages. Since the relation is ordered on salary and from a) it is known 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.

Strategy 2 page I/O cost: 3 + 25 = 28

Step 1 page I/O cost: 28

From a) 2,000 tuples are expected to be retrieved. Only empId and deptId are retained which will occupy $\lceil 2000 / \lfloor 4000 / 8 \rfloor \rceil = 4$ pages and which are kept in the buffer to join with Department in the next step.

Step 1 output size (tuples): 2,000

Step 1 output size (pages): 4

```
select distinct empId, deptId, projectId
from Employee natural join Department
      natural join DeptProject
      natural join Project
where salary=>100000
      and budget>20000;
```

EXERCISE 3 (CONT'D)

$n_{Employee}$: 20,000 tuples
$B_{Employee}$: 250 pages
$n_{Department}$: 100 tuples
$B_{Department}$: 1 pages
$n_{DeptProject}$: 4,000
$B_{DeptProject}$: 8 pages
$n_{Project}$: 1,000 tuples
$B_{Project}$: 500 pages
Page size: 4,000 bytes
M: 12 pages
B^+-tree: Employee.salary
Hash index: Department.deptId
Hash index: Project.projectId

Step 2: result A ⋈ Department ⟹ result B

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

For each of the 2,000 Step 1 tuples, 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.

Strategy 1 page I/O cost: 2000 * 2 = 4000

Strategy 2: block nested-loop join

The Department relation occupies $\lceil 100 / \lfloor 4000/36 \rfloor \rceil = 1$ page. This page can be read into memory and joined with a tuple from the previous step using block nested loop join.

Strategy 2 page I/O cost: 1

Step 2 page I/O cost: 1

Each of the 2,000 Step 1 tuples is expected to match with a Department tuple. Retain only empId and deptId which will occupy $\lceil 2000 / \lfloor 4000 / 8 \rfloor \rceil = 4$ pages and which are kept in the buffer to join with DeptProject.

Step 2 output size (tuples): 2,000 **Step 2 output size (pages): 4**

```
select distinct empId, deptId, projectId
from Employee natural join Department
        natural join DeptProject
        natural join Project
where salary=>100000
         and budget>20000;
```

EXERCISE 3 (CONT'D)

$n_{Employee}$: 20,000 tuples
$B_{Employee}$: 250 pages
$n_{Department}$: 100 tuples
$B_{Department}$: 1 pages
$n_{DeptProject}$: 4,000
$B_{DeptProject}$: 8 pages
$n_{Project}$: 1,000 tuples
$B_{Project}$: 500 pages
Page size: 4,000 bytes
M: 12 pages
B^+-tree: Employee.salary
Hash index: Department.deptId
Hash index: Project.projectId

Step 3: result B ⋈ DeptProject ⟹ result C

Strategy: block nested-loop join

Since there is no index on DeptProject, block nested-loop is the only possible strategy for the join. The cost is that to read the DeptProject relation.

Strategy page I/O cost: 8

As estimated in a) each of the 2,000 tuples from Step 2 is expected to match with 40 DeptProject tuples. Thus, the expected output size is 80,000 tuples. Retain only empId, deptId and projectId which will occupy ⌈80,000 / ⌊4000 / 12⌋⌉ = 241 pages. The join result cannot be kept in the buffer but needs to be written to disk requiring 241 page I/Os.

Join result write page I/O cost: 241

Step 3 page I/O cost: 8 + 241 = 249

Step 3 output size (tuples): 80,000

Step 3 output size (pages): 241

Step 4: $\sigma_{budget>20000}$Project ⟹ result D

Strategy: do on-the-fly after join

Step 4 page I/O cost: 0

```
select distinct empId, deptId, projectId
from Employee natural join Department
        natural join DeptProject
        natural join Project
where salary=>100000
        and budget>20000;
```

EXERCISE 3 (CONT'D)

$n_{Employee}$: 20,000 tuples
$B_{Employee}$: 250 pages
$n_{Department}$: 100 tuples
$B_{Department}$: 1 pages
$n_{DeptProject}$: 4,000
$B_{DeptProject}$: 8 pages
$n_{Project}$: 1,000 tuples
$B_{Project}$: 500 pages
Page size: 4,000 bytes
M: 12 pages
B⁺-tree: Employee.salary
Hash index: Department.deptId
Hash index: Project.projectId

Step 5: result C $\bowtie$ result D ≡ result C $\bowtie$ $\sigma_{budget>20000}$Project

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

For the 80,000 Step 3 tuples, use the projectId hash index on Project to join 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 80,000 tuples.

Strategy 1 page I/O cost: 80,000 * 2 = 160,000

Strategy 2: optimized block nested-loop join

Use optimized block nested-loop join with Result C as the outer relation.

Strategy 2 page I/O cost: $\lceil 241 / (12-2) \rceil * 500 + 241 = 12{,}741$

Step 5 page I/O cost: 12,741

As estimated in a) every Result C tuple will join with a Project tuple, but only 0.4 of these tuples will meet the condition $\sigma_{budget>20000}$. Thus, the estimated output size is 80,000 * 0.4 = 32,000 tuples or $\lceil 32{,}000 / \lfloor 4000 / 12 \rfloor \rceil = 97$ pages.

Step 5 output size (tuples): 32,000   Step 5 output size (pages): 97

Query processing page I/O cost: 28 + 1 + 249 + 0 + 12,741 = 13,019

Query result size (pages): 97